Universidad de Navarra

MASTER EN BIG DATA SCIENCE.

Análisis de cotizaciones en Kraken.

Python para análisis de datos.

Agustín Morales Córdoba.

Lucía Colín Cosano.

2022-2023

ÍNDICE

1. INTRODUCCIÓN.

El objetivo del proyecto es ser capaz de poder obtener cotizaciones de criptomonedas utilizando Kraken. Kraken es una página que sirve para el exchange de criptomonedas. Esta se encarga de emparejar órdenes de clientes que buscan comprar con las de clientes que desean vender.

Se popularidad nace en 2014 debido a que fue elegida para ofrecer datos de comercio de Bitcoin de forma que los clientes pudiesen acceder a sus precios, gráficos y noticias relacionadas.

Gracias a la información tan valiosa que esta página ofrecía, se desarrolló una librería en Python, **Krakenex**, que permite la extracción de datos a través de una API.

En cuanto a como se ha hecho frente al proyecto, en primer lugar se han analizado los objetivos y como hacer frente a ellos. Una vez entendidos se ha desarrollado el código siguiendo los pasos dados en el *notebook* y tras ello se ha mejorado y optimizado utilizando funciones y clases. Finalmente se ha implementado el manejo de errores y una vez finalizado el proyecto este ha sido subido a GitHub.

2. LECTURA Y REPRESENTACIÓN DEL MOVIMIENTO DEL PAR DE MONEDAS.

Para la lectura de los datos se ha utilizado la librería de Krakenex. Ademas se han implementado las librerías necesarias para el tratamiento de los datos y la representación gráfica de los resultados obtenidos.

2.1. DESCARGA DE DATOS UTILIZANDO LA LIBRERIA KRAKEN.

Para conectar con la API de Kraken, se ha definido una función que permite esta conexión. En este caso se está utilizando la versión gratuita por lo que el numero de datos que se va a poder extraer de una sola consulta es limitado.

Una vez establecida la conexión, se utiliza la función $get\_asset\_info()$, que permite obtener los activos disponibles. Esta información se almacena en forma de DataFrame.

Se define entonces otra función que permite acceder a las diferentes criptomonedas disponibles a través de la función $get\_tradable\_asset\_pairs()$ dentro de la función $getpairs$.

```
def conectarKrakenAPI():
  try:
    api = krakenex.API('api-key-1668535705986')
    k = KrakenAPI(api)

    data = k.get_asset_info()
    data=pd.DataFrame(data)
  except:
    print("No ha sido posible establecer la conexión")
  return k

def getpairs(self):
  k=self.conectarKrakenAPI()
  pairs = k.get_tradable_asset_pairs()
  pairs.head().T
  pairs=pd.DataFrame(pairs)
  return pairs
```

Llegados al punto en el que se conoce la información existente, se procede a definir una función que permita extraer dichos datos. Para ello se define una función, que tiene tres argumentos: la moneda deseada, la fecha desde la que se quieren obtener los datos y el nombre del archivo formato *csv* en el que se quiere guardar la base de datos. Estos argumentos serán definidos por el usuario a través de inputs.

El formato de la fecha no es óptimo para las funciones de obtención de datos existentes, por lo que se define un bucle en el que este se transforma en un objeto de tipo datetime. Una vez tratada, se conecta con la API de Kraken y se obtienen los datos con la función $get\_ohlc\_data(pair = "", interval = 1, since = 0)$, donde pair es la criptomoneda elegida, el intervalo define el número de minutos que existe entre cada dato consecutivo y finalmente since es la fecha desde la que se desea extraer datos. Cabe destacar que *since* solo funciona para fechas comprendidas dentro del rango de datos accesibles con l versión gratuita.

En este caso se ha decidido obtener datos diarios y, debido a la limitación de la versión gratuita utilizada, el número de datos disponibles por cada consulta corresponde a 720 días.

```
    def sacardatos(self,moneda, fecha, nombre):
    try:
      d=fecha.split("-")
      for x in range(len(d)):
        d[x]=int(d[x])
      date_time_inicio=datetime.date(d[0],d[1],d[2])
      datex=date_time_inicio.timetuple()
      date_inicio=time.mktime(datex)
    except SyntaxError:
      print("La fecha introducida no es valida y por tanto no se puede ejecutar
    ↪ la transformacion")

    conect=self.conectarKrakenAPI()
    datos = conect.get_ohlc_data(moneda, interval=1440, since=date_inicio,
    ↪ ascending = True)
    datos=datos[0]
    datos=pd.DataFrame(datos)

    x=nombre+'.csv'
    datos.to_csv(x, header=True, index=False)

    return datos
```

2.2. GRÁFICOS DE COTIZACIONES.

Se define una función que permite obtener una gráfica que contiene los datos de cierre, apertura, máximo y mínimo. La elección de la moneda se encuentra dentro del argumento de la función, *datos*.

```
def graficarmonedas(datos):
    fig, ax = plt.subplots()
    plt.title('ANÁLISIS DEL ' + moneda ,fontname='Times New Roman',
    ↪ fontweight='bold')
    plt.xlabel("Fecha",fontname='Times New Roman', fontweight='bold',fontsize=2)
    plt.ylabel('Precio en euros',fontname='Times New Roman', fontweight='bold')
    ax.grid(linestyle='dotted', linewidth=0.6)
    ax.tick_params(axis='x', rotation=60)
    ax.set_facecolor("ivory")

    plt.plot(datos['open'],'teal',label='Apertura')
    plt.plot(datos['close'],'SteelBlue',label='Cierre')
    plt.plot(datos['high'],'springgreen',label='Máximo')
    plt.plot(datos['low'],'darkred',label='Mínimo')
    plt.legend(loc='best', facecolor='w', fontsize=9)
    fig.set_size_inches(12, 6)
    plt.savefig('infomoneda'+moneda,dpi=150, bbox_inches='tight')
    return plt.show()
```

2.2.1. GRAFICAR EL PAR ETHUSD.

Utilizando la función anterior se obtiene la gráfica del par *ETHUSD*.

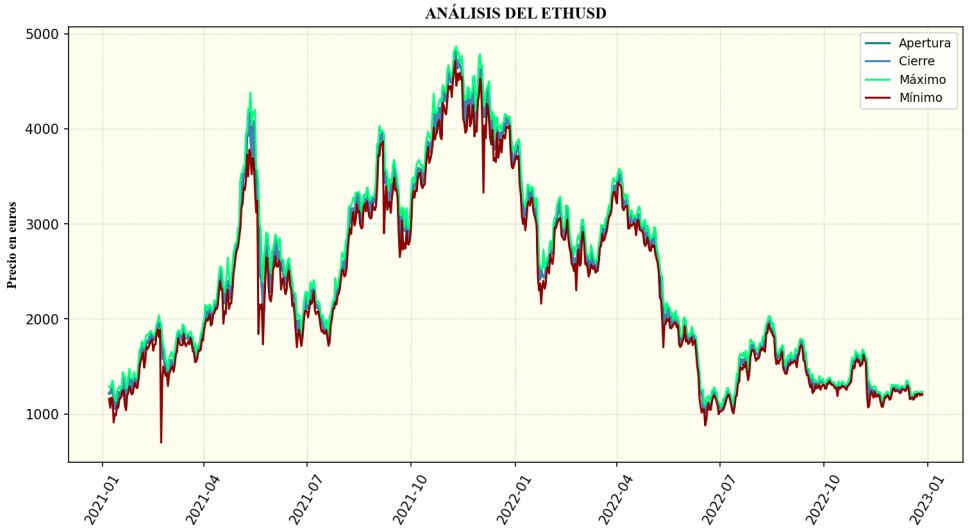

Figura 2.1: Gráfica de la cotización del par ETHUSD.

2.2.2. INPUT DE USUARIO QUE PERMITA GRAFICAR UNA COTIZACIÓN A ELEGIR.

Se pide al usuario que introduzca la moneda que desea graficar. Una vez insertada se comprueba que esta es una moneda válida, asegurando que se encuentra en la lista disponible (*pairs*).

Una vez se tiene la moneda, esta se utiliza en la función *sacardatos* para obtener los datos correspondientes y con la función *graficarmonedas* se tiene el resultado esperado.

```
1     moneda=input("Introduzca la moneda de la que desea obtener las
  ↪ cotizaciones:")
2     fecha=input("Escribir desde que dia se quieren los datos poniendo primero
  ↪ el año, luego el mes y luego el dia, separado con guiones ")
3     nombre=input("Nombre del csv en el que se guarda la información: ")
4
5     obtencion_de_datos=obtencion_de_datos()
6
7     pairs=obtencion_de_datos.getpairs()
8     if moneda in list(pairs['altname']):
9       moneda=moneda
10    else:
11      print("La moneda introducida no existe")
12      moneda= input("Introduzca otra moneda:")
13
14    datos=obtencion_de_datos.sacardatos(moneda, fecha, nombre)
```

2.2.3. OTROS GRÁFICOS DE INTERÉS.

A continuación, se muestran otros gráficos que se consideran de interés para el estudio de las cotizaciones.

En primer lugar, se representa en un gráfico conjunto el valor de cierre y de *vwap*4. El *vwap* es el premio medio ponderado por volumen. Este es utilizado para conocer el precio medio de un determinado activo en un periodo de tiempo determinado, y por tanto sirve como referencia de negociación para los inversores.

Los inversores pasivos son sus usuarios habituales ya que estos derivan sus acciones a programas de informativos y esta métrica es una buena referencia.

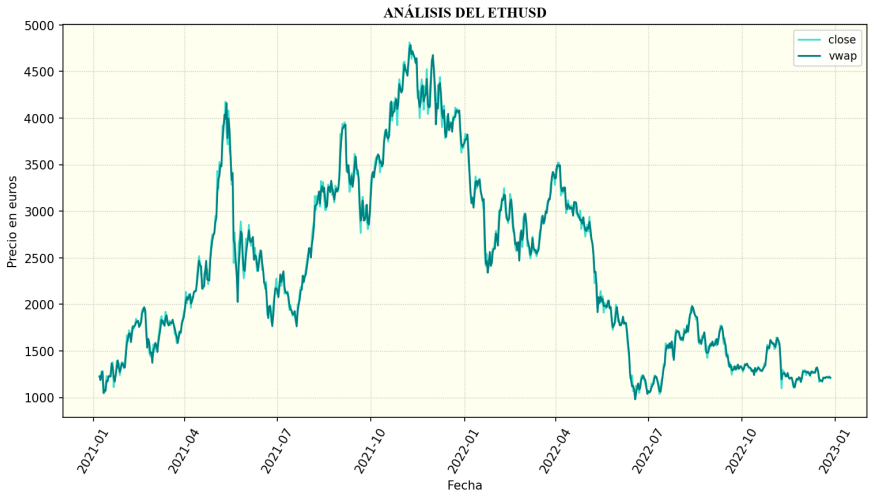

Figura 2.2: Gráfica conjunta del valor de cierre y vwap.

Se grafica un gráfico de velas japones con el fin de mostrar información sobre las variaciones de precio que sufre la criptomoneda en cada momento. En ellos destaca la importancia de los colores utilizados ya que ofrecen una rápida imagen visual.

Figura 2.3: Gráfico de velas japonesas del Ethereum.

En la imagen que se muestra a continuación se grafica las velas japonesas para un periodo mas reducido y poder así apreciarlo mejor.

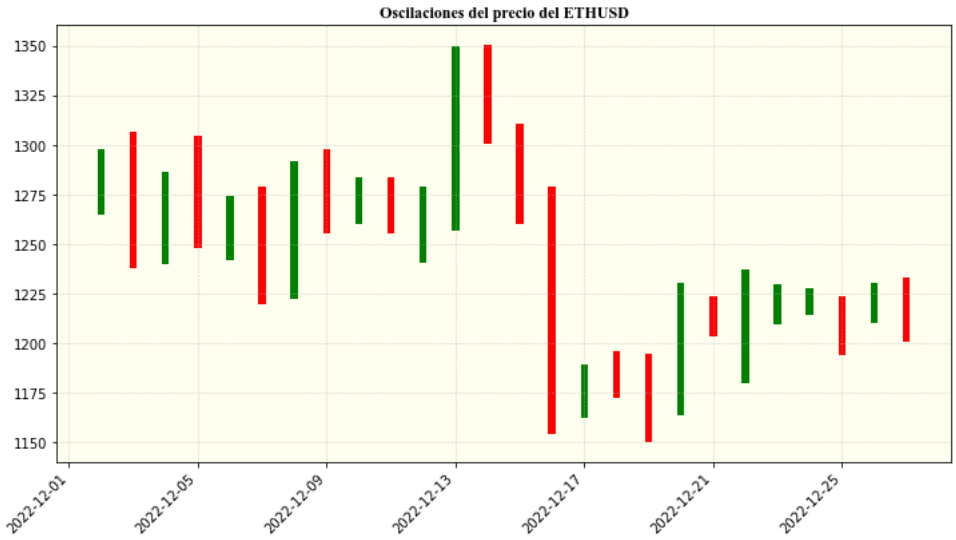

Figura 2.4: Gráfico de velas japonesas del Ethereum.

El código correspondiente a dicho gráfico se adjunta a continuación. En el se filtran por serados aquellos días en los que el valor de cierre es mayor y viceversa. A partir de esta clasificación se grafican utilizando diferentes colores.

Se han omitido las lineas de código correspondientes a la personalización de la gráfica.

```
def candlestick(datos):
  fig, ax = plt.subplots()
  plt.title('Oscilaciones del modelo del ' + moneda,fontname='Times New Roman',
    ↪ fontweight='bold' )

  up = datos[datos.close>=datos.open]
  down = datos[datos.close<datos.open]

  plt.bar(up.index,up.close-up.open,width,bottom=up.open,color='green')
  plt.bar(up.index,up.high-up.close,width2,bottom=up.close,color='green')
  plt.bar(up.index,up.low-up.open,width2,bottom=up.open,color='green')

  plt.bar(down.index,down.close-down.open,width,bottom=down.open,color='red')
  plt.bar(down.index,down.high-down.open,width2,bottom=down.open,color='red')
  plt.bar(down.index,down.low-down.close,width2,bottom=down.close,color='red')

  plt.savefig('candle'+moneda,dpi=150, bbox_inches='tight')
  return plt.show()
```

3. INDICADORES TÉCNICOS.

3.1. CÁLCULO DE LA MEDIA MÓVIL Y GRÁFICO.

La media móvil es un indicador que permite combinar valores a lo largo de un tiempo determinado para calcular sus precios medias al dividirlo entre el numero de datos recogidos para poder desarrollar una linea de tendencia. Su interés fundamental es detectar la dirección que tomará una tendencia y por tanto reducir así su impacto.

La media móvil se calcula sobre el precio de cierre. Es muy sensible a grandes variaciones de este valor. Su expresión se muestra a continuación.

$$f(x) = \frac{\sum_{i=1}^{nsesiones} Preciodecierre}{nsesiones} \tag{3.1}$$

Para el calculo de la media móvil se toman los datos de salida y se le da la vuelta a x para posteriormente implementar dos bucles *for* anidados, donde nsesiones es numero de periodos a utilizar. El invertir los datos es necesario para el bucle *for* ya que para los primeros *nsesiones* no existe un valor de la media móvil correspondiente.

```
def mediamovilsimple(nsesiones):
  x=datos['close']
  x=x[::-1]
  mediamovil=[]
  for i in range(0,len(datos['close'])-nsesiones):
    suma=0
    for j in range(i,i+nsesiones):
      suma=datos['close'][j]+suma
    mms=(suma/nsesiones)
    mediamovil.append(mms)
  return mediamovil
```

Por lo general la media móvil a corto plazo $nsesions \subset (0, 20)$ se utilizan para determinar los impulsos mientras que las de medio plazo sirven para determinar las zonas de corrección.

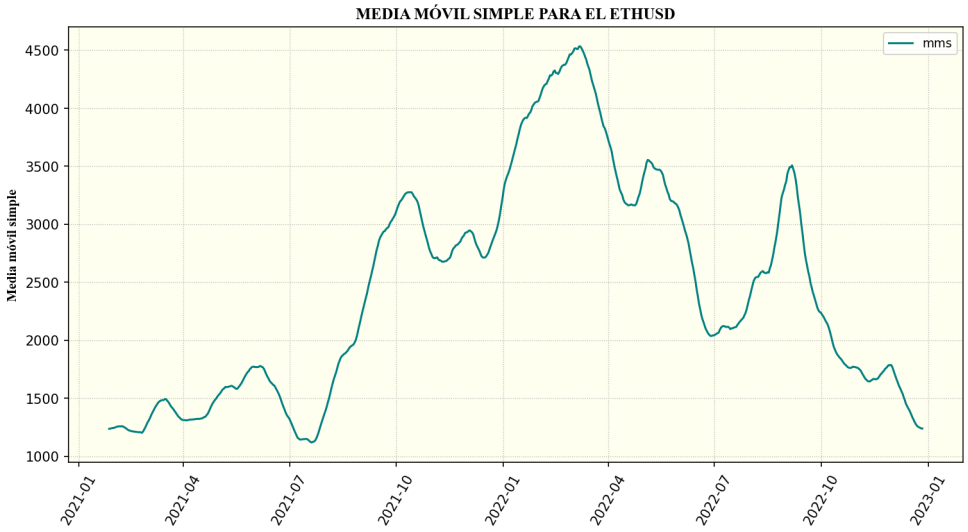

Figura 3.1: Media móvil y valor de cierre para nsesions=20.

3.2. CÁLCULO DEL RSI Y GRÁFICO.

A continuación, se procede a calcular el RSI. Para ello es necesario calcular previamente la media móvil exponencial. Esta es comúnmente utilizada ya que otorga mas importancia a los datos mas recientes y, aunque es mas sensible, ofrece una visión mas realista de la situación analizada.

```
    def mediamovilexponencial(nsesiones):
    multiplicador=2/(nsesiones+1)
    mediamovilexp=[]
    longitud=len(mediamovilsimple(nsesiones))
    for j in range(0,longitud):
    suma=0
    suma=multiplicador*(mediamovilsimple(nsesiones)[j-1])+
    (mediamovilsimple(nsesiones)[j-1])+suma
    mme=(suma)
    mediamovilexp.append(mme)
    return mediamovilexp
```

El RSI o Relative Strength Index es un marcador utilizado para analizar la fuerza con la que suben y bajan los precios de cierre de, en este caso, las criptomonedas. Para su calculo es necesario detectar los periodos alcistas y los periodos bajistas. Este se calcula a partir de la fuerza relativa inicial cuya expresión queda definida por la media móvil exponencial multiplicada por el número de periodos alcistas entre la media móvil exponencial del numero de periodos bajistas. Una vez obtenido ese valor el RSI queda definido por:

$$f(x) = 100 - \frac{100}{1 + RS} \tag{3.2}$$

Es importante interpretar este coeficiente ya que aporta información de calidad. Este valor oscila entre 0 y 100 y se considera zona neutra la comprendida entre 30 y 70.

```
    def rsi(datos, periods = 14, ema = True):

    close_diff= datos['close'].diff()

    up = close_diff.clip(lower=0)
    down = -1 * close_diff.clip(upper=0)

    if ema == True:
    ma_up = up.ewm(com = periods - 1, adjust=True, min_periods =
↪ periods).mean()
    ma_down = down.ewm(com = periods - 1, adjust=True, min_periods =
↪ periods).mean()
    else:
    ma_up = up.rolling(window = periods, adjust=False).mean()
    ma_down = down.rolling(window = periods, adjust=False).mean()

    rsi = ma_up / ma_down
    rsi = 100 - (100/(1 + rsi))
    return rsi
```

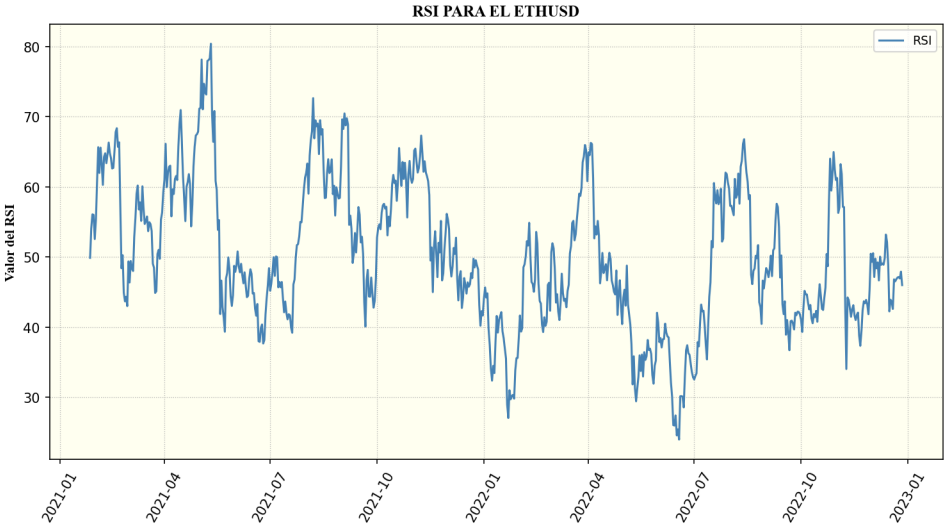

Figura 3.2: Relative Strength Index (RSI) para el Ethereum.

3.3. GRÁFICA DE LA MEDIA MÓVIL JUNTO A LA COTIZACIÓN DEL PAR CORRESPONDIENTE.

A continuación se muestra la gráfica correspondiente a $nsesions = 3$ y $nsesions = 20$.

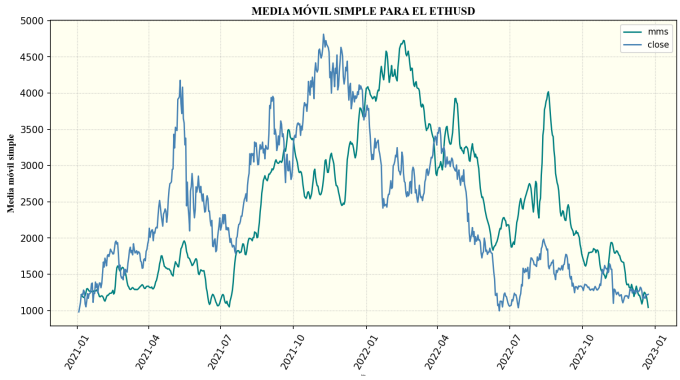

Figura 3.3: Media móvil y cotización del par considerando 3 periodos.

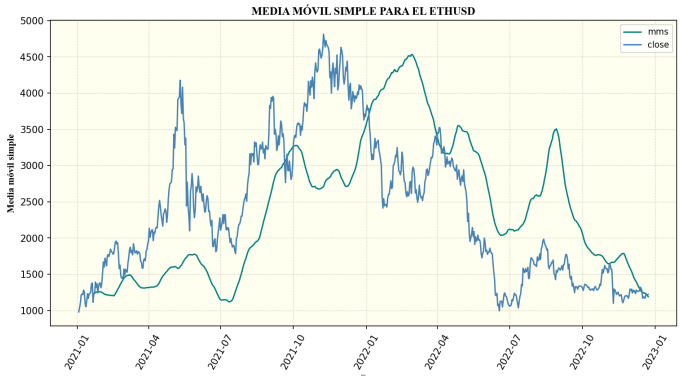

Figura 3.4: Media móvil y cotización del par considerando 20 periodos.

4. ESTRUCTURACIÓN DEL CÓDIGO.

El código se ha desarrollado en *spyder*.

En primer lugar se han definido las funciones siguiendo el orden lógico de ejecución y finalmente se han implementado en el programa principal. Ademas se ha creado una clase para la obtención de datos que se adjunta a continuación y otra sobre gráficas.

En cuanto al manejo de errores y excepciones, en el código adjuntado se puede comprobar su implementación.

```
class obtencion_de_datos:
  def __post_init__(self,moneda,fecha,nombre):
    self.moneda = moneda
    self.fecha = fecha
    self.nombre = nombre

  def conectarKrakenAPI(self):
    try:
      api = krakenex.API('api-key-1668535705986')
      k = KrakenAPI(api)

      data = k.get_asset_info()
      data=pd.DataFrame(data)
    except:
      print("No ha sido posible establecer la conexión")
    return k

  def getpairs(self):
    k=self.conectarKrakenAPI()
    pairs = k.get_tradable_asset_pairs()
    pairs.head().T
    pairs=pd.DataFrame(pairs)
    return pairs

  def sacardatos(self,moneda, fecha, nombre):
    try:
      d=fecha.split("-")
      for x in range(len(d)):
        d[x]=int(d[x])
      date_time_inicio=datetime.date(d[0],d[1],d[2])
      datex=date_time_inicio.timetuple()
      date_inicio=time.mktime(datex)
    except SyntaxError:
      print("La fecha introducida no es valida y por tanto no se puede ejecutar
    ↪ la transformacion")

    conect=self.conectarKrakenAPI()
    datos = conect.get_ohlc_data(moneda, interval=1440, since=date_inicio,
    ↪ ascending = True)
    datos=datos[0]
    datos=pd.DataFrame(datos)

    x=nombre+'.csv'
    datos.to_csv(x, header=True, index=False)
    return datos
```

5. DISTRIBUCIÓN DEL CÓDIGO.

El código ha sido subido a Github. Se adjunta el enlace al repositorio.

https://github.com/luciacolin/Trabajo-Python.git

6. APLICACIÓN AL BITCOIN.

A continuación se adjuntas las gráficas correspondientes al Bitcoin. Para obtenerlas para otra moneda cualquiera solo es necesario introducir su abreviatura al ejecutar el programa.

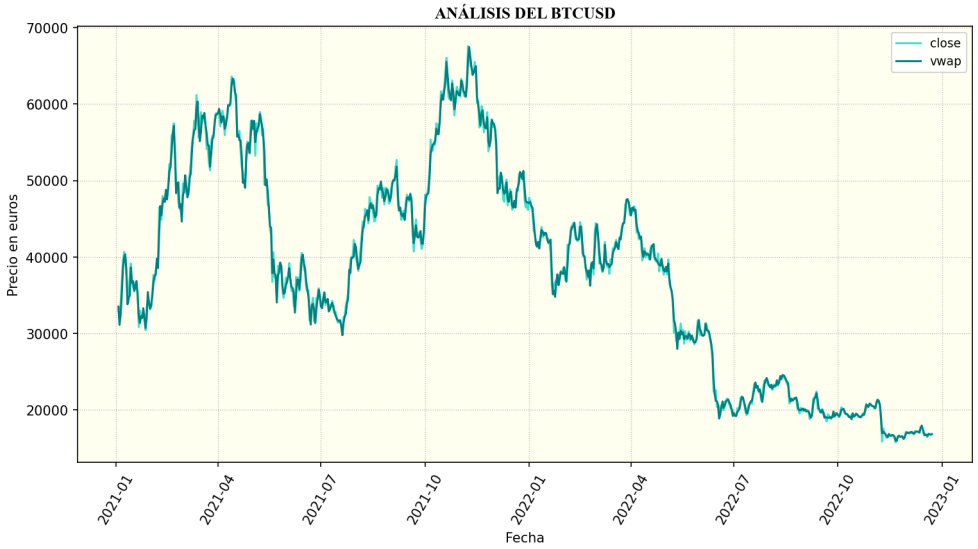

Figura 6.1: Gráfica conjunta del valor de cierre y vwap.

Figura 6.2: Gráfico de velas japonesas del Bitcoin.

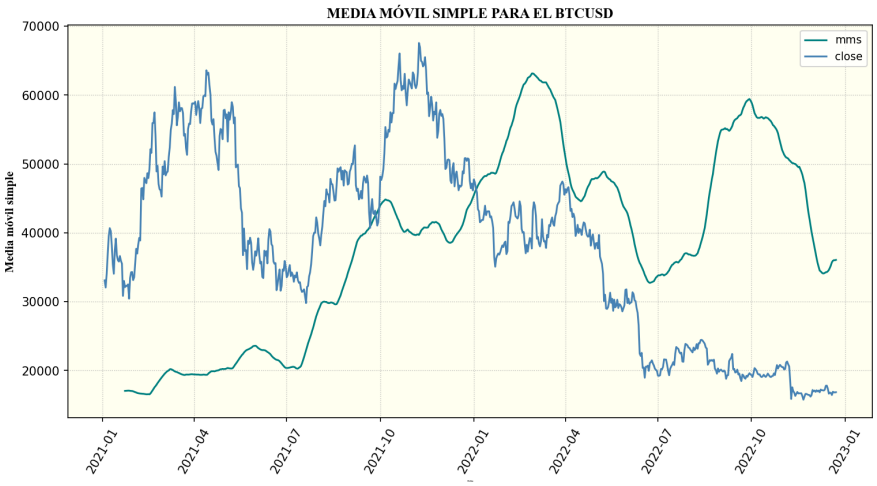

Figura 6.3: Media móvil y valor de cierre para nsesions=20.

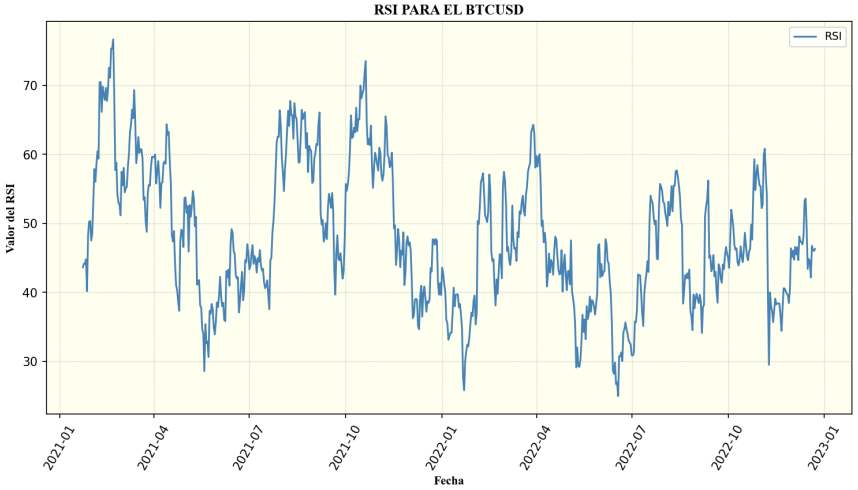

Figura 6.4: Relative Strength Index (RSI) para el Bitcoin.

ÍNDICE DE FIGURAS