



**Universidad
de Navarra**

DATAI
INSTITUTO DE CIENCIA DE LOS
DATOS E INTELIGENCIA ARTIFICIAL

Proyecto Final para la asignatura Python para Análisis de Datos

Máster Universitario en Big Data Science

Curso académico 2023-2024

Guillermo Martín Villaverde

Pablo Legerén Somolinos e Iker Yáñez Calderón



1. Enfoque.....	2
1.1 Control de Versiones	2
1.2 Integración Continua	2
1.3 Programación Orientada a Objetos (POO)	2
1.4 Aplicación Web	2
2. Componentes evaluables	3
2.1 Lectura y representación del movimiento del par de monedas	3
2.1.1 Descarga de datos	3
2.1.2 Graficar cotizaciones	3
2.2 Indicadores Técnicos	6
2.3 Estructuración	9
3. Ejecución.....	11
4. Extra	12
4.1 Testing.....	12
4.2 Reproducción del entorno	13
4.3 Distribución	13



1. Enfoque

El desarrollo de este proyecto ha sido guiado por un enfoque metodológico que destaca la importancia de la colaboración eficiente, la integración continua y la estructuración organizada del código.

1.1 Control de Versiones

Para gestionar eficazmente las contribuciones y mantener un historial claro de cambios, el uso de un sistema de control de versiones se volvió esencial. Git ha sido la piedra angular en este aspecto, permitiéndonos trabajar de manera concurrente en diferentes partes del proyecto, fusionar sus cambios de manera eficiente y revertir modificaciones si fuera necesario. La implementación de un flujo de trabajo basado en Git ha mejorado significativamente la colaboración y la trazabilidad del código.

1.2 Integración Continua

La integración continua (CI) ha sido un componente vital de nuestro proceso de desarrollo. Hemos optado por GitHub Actions, una plataforma de CI/CD que automatiza la ejecución de pruebas y flujos de trabajo predefinidos en cada confirmación de código. Esta práctica garantiza que cualquier cambio introducido en el repositorio se someta a una verificación automática, por lo que cualquier error en la ejecución principal o de los tests se resaltaba.

1.3 Programación Orientada a Objetos (POO)

La organización efectiva del código ha sido una prioridad para facilitar su uso y comprensión. Hemos optado por la Programación Orientada a Objetos (POO) para estructurar el código. Esta elección ha facilitado la mantenibilidad, la extensibilidad y la comprensión del código. Cada componente del proyecto, como el menú de selección, el descargador de datos, los indicadores y los gráficos, se ha modelado como una clase independiente, promoviendo así una arquitectura de código limpia y modular.

1.4 Aplicación Web

Cabe destacar que el proyecto culmina en una aplicación web gracias a la librería Streamlit, por lo que a lo largo de la memoria se mostrarán evidencias del resultado final en la web.



2. Componentes evaluables

2.1 Lectura y representación del movimiento del par de monedas

2.1.1 Descarga de datos

Para llevar a cabo la descarga de datos, hemos seleccionado la librería de Kraken para Python como nuestra herramienta principal. Con el propósito de organizar y encapsular esta funcionalidad, hemos creado una clase dedicada que incluye una función específica para la descarga del conjunto de datos asociado al par de monedas proporcionado como parámetro. Más adelante, se detallará cómo el usuario puede especificar dicho par de monedas.

```
DescargadorDatos.py x
1 import krakenex
2 import pandas as pd
3 from pykrakenapi import KrakenAPI
4
5 6 usages  Iker Yañez *
6 class DescargadorDatos:
7     Iker Yañez
8     def __init__(self):
9         self.api = krakenex.API()
10        self.kraken_api = KrakenAPI(self.api)
11
12    2 usages  Iker Yañez *
13    def descargar_datos(self, par):
14        try:
15            ohlc, last = self.kraken_api.get_ohlc_data(par, ascending=True)
16            ohlc = pd.DataFrame(ohlc)
17            ohlc[['open', 'high', 'low', 'close']] = ohlc[['open', 'high', 'low', 'close']].astype(float)
18            return ohlc
19        except Exception as e:
20            print(f"Error al descargar datos: {e}")
21            return None
```

2.1.2 Graficar cotizaciones

En primer lugar, hemos seleccionado las diez criptomonedas con mayor capitalización de mercado para incluirlas en un desplegable que permita al usuario seleccionar la que desee. Para ello hemos creado una clase independiente.



```
DescargadorDatos.py  Graficos.py  Menu.py x
1  from DescargadorDatos import DescargadorDatos
2  import streamlit as st
3
4  class Menu:
5      def __init__(self):
6          self.descargador = DescargadorDatos()
7          self.pares = {
8              1: "BTC/USD",
9              2: "ETH/USD",
10             3: "USDT/USD",
11             4: "XRP/USD",
12             5: "USDC/USD",
13             6: "SOL/USD",
14             7: "ADA/USD",
15             8: "DOGE/USD",
16             9: "TRX/USD",
17             10: "LINK/USD",
18         }
19
20     def menu(self):
21         st.sidebar.header("Selección de Par")
22
23         # Selección de par de monedas
24         opcion_par = st.sidebar.selectbox("Seleccione un par de monedas:", list(self.pares.values()))
25
26         return opcion_par
```

Su función principal devuelve el par seleccionado por el usuario, sirviendo como entrada para el descargador de datos. En la web, el menú se aprecia de la siguiente forma:



Para las funciones relacionadas con los gráficos también hemos decidido crear una clase específica. Por el momento, tan solo hablaremos de la función que muestra la cotización del par seleccionado. Esta función tiene como objetivo generar y mostrar un gráfico interactivo de velas utilizando la biblioteca Plotly.

```
DescargadorDatos.py  Graficos.py x
1 import streamlit as st
2 import plotly.graph_objects as go
3 import plotly.express as px
4
5 class Graficos:
6
7     def __init__(self, datos):
8         self.datos = datos
9
10    def graficar_datos(self):
11        # Gráfico de velas con Plotly
12        st.header("Gráfico de Velas")
13        fig = go.Figure(data=[go.Candlestick(x=self.datos.index,
14                                             open=self.datos['open'],
15                                             high=self.datos['high'],
16                                             low=self.datos['low'],
17                                             close=self.datos['close'])])
18        st.plotly_chart(fig)
```

Eventualmente, resulta en un gráfico con la siguiente apariencia:



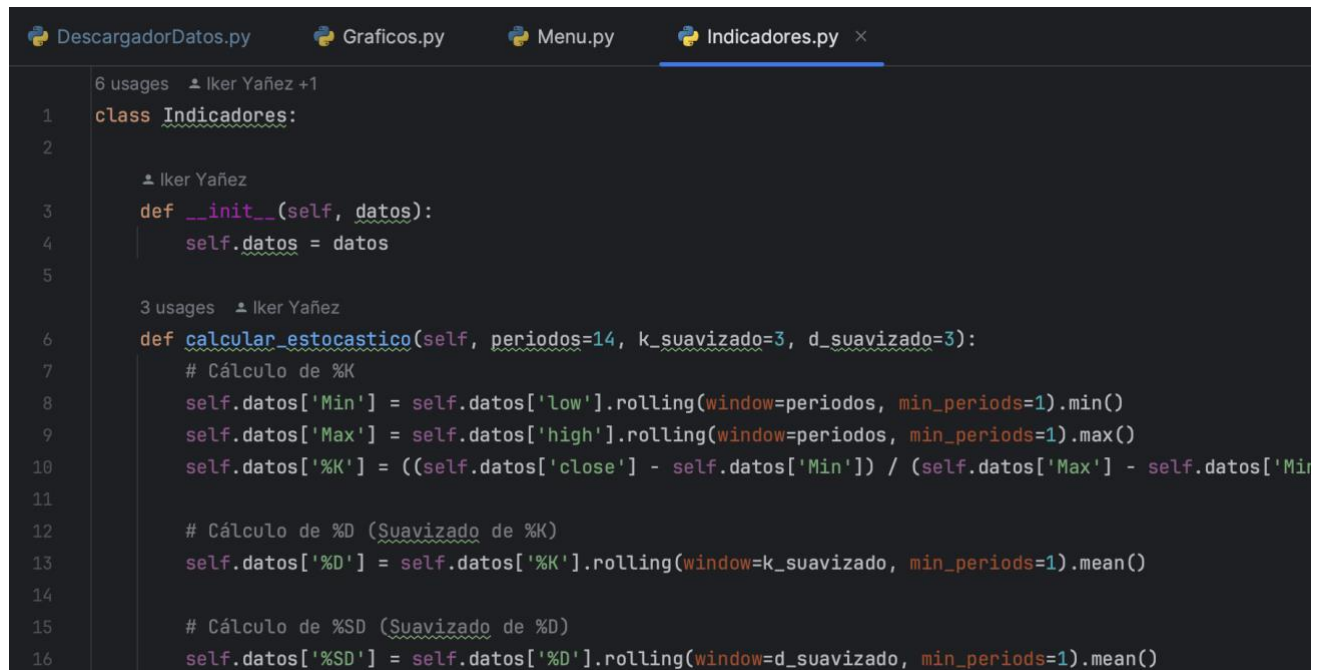
Además, contamos con un filtro en la parte inferior y la posibilidad de navegar libremente por el gráfico. Por ejemplo, hacer zoom o estirar el eje temporal para mayor detalle.



2.2 Indicadores Técnicos

En este caso se cuenta con la clase Indicadores para el cálculo de los estocásticos, y para graficarlos se recurrirá a la anteriormente mostrada clase Graficos.

En primer lugar, se muestra la función que realiza el cálculo de los estocásticos:



```
DescargadorDatos.py Graficos.py Menu.py Indicadores.py x
6 usages  Iker Yañez +1
1 class Indicadores:
2
3     Iker Yañez
4     def __init__(self, datos):
5         self.datos = datos
6
7     3 usages  Iker Yañez
8     def calcular_estocastico(self, periodos=14, k_suavizado=3, d_suavizado=3):
9         # Cálculo de %K
10        self.datos['Min'] = self.datos['low'].rolling(window=periodos, min_periods=1).min()
11        self.datos['Max'] = self.datos['high'].rolling(window=periodos, min_periods=1).max()
12        self.datos['%K'] = ((self.datos['close'] - self.datos['Min']) / (self.datos['Max'] - self.datos['Min'])) * 100
13
14        # Cálculo de %D (Suavizado de %K)
15        self.datos['%D'] = self.datos['%K'].rolling(window=k_suavizado, min_periods=1).mean()
16
17        # Cálculo de %SD (Suavizado de %D)
18        self.datos['%SD'] = self.datos['%D'].rolling(window=d_suavizado, min_periods=1).mean()
```

Esta función se encarga de calcular los indicadores estocásticos (%K, %D, %SD) para un conjunto de datos dado. Al aplicar esta función al dataframe, se agregan nuevas columnas que representan estos indicadores calculados. En términos generales, el proceso implica:

- Cálculo de %K: Se determina el mínimo (Min) y el máximo (Max) de los precios más bajos (low) y más altos (high) en un periodo específico. Se utiliza la fórmula estocástica para calcular %K basándose en el precio de cierre (close), el mínimo y el máximo. El resultado se almacena en una nueva columna %K.
- Suavizado de %K para obtener %D: Se aplica un suavizado a la columna %K mediante una media móvil definida por el parámetro k_suavizado. El resultado se almacena en una nueva columna %D.
- Suavizado Adicional para %SD: Se realiza un suavizado adicional a la columna %D mediante una media móvil definida por el parámetro d_suavizado. El resultado se almacena en una nueva columna %SD.

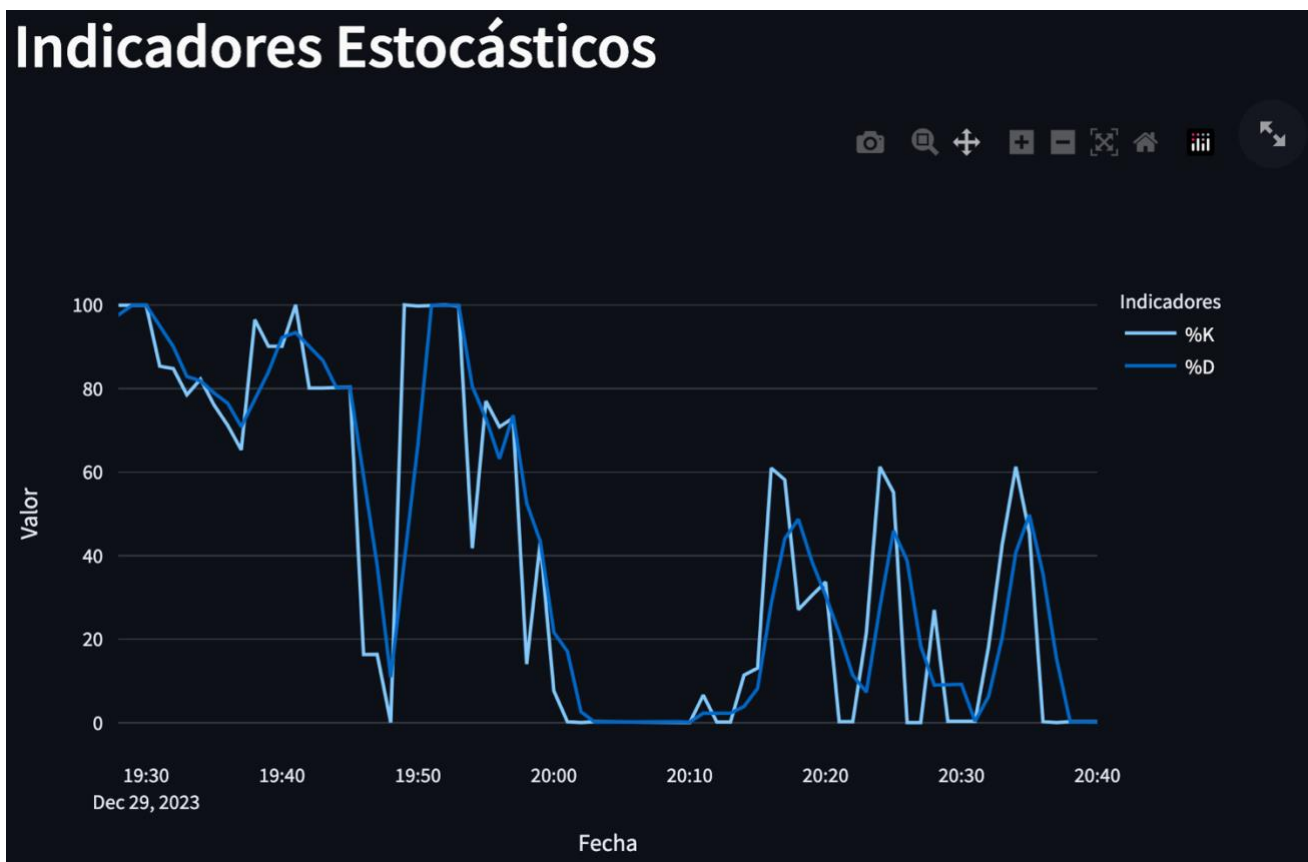
A continuación, se muestra la función que realiza el grafico de los estocásticos:



```
DescargadorDatos.py Graficos.py x Menu.py Indicadores.py

1 usage  Iker Yañez *
20 def graficar_estocastico(self):
21     # Gráfico de %K y %D
22     st.header("Indicadores Estocásticos")
23     fig_estocastico = go.Figure()
24
25     fig_estocastico.add_trace(go.Scatter(x=self.datos.index, y=self.datos['%K'], mode='lines', name='%K'))
26     fig_estocastico.add_trace(go.Scatter(x=self.datos.index, y=self.datos['%D'], mode='lines', name='%D'))
27
28     # Configura el diseño del gráfico
29     fig_estocastico.update_layout(
30         xaxis_title="Fecha",
31         yaxis_title="Valor",
32         legend=dict(title="Indicadores"),
33     )
34
35     # Muestra el gráfico de Plotly
36     st.plotly_chart(fig_estocastico, use_container_width=True, width=0)
37
```

Cabe destacar que solo se muestran %K y %D, que son los indicadores en los que nos vamos a basar para realizar una función adicional, mostrar las órdenes de compra y venta. El gráfico de los indicadores en la web se muestra de la siguiente forma:



Como se ha mencionado anteriormente, hemos aprovechado los indicadores para realizar el cálculo de órdenes de compra y venta, gracias a la siguiente función de la clase Indicadores:



2 usages ▴ Pablo

```
def calcular_ordenes(self):
    condiciones_compra = ((self.datos['%K'] > self.datos['%D']) &
                          (self.datos['%K'].shift(1) <= self.datos['%D'].shift(1)))
    condiciones_venta = ((self.datos['%K'] < self.datos['%D']) &
                        (self.datos['%K'].shift(1) >= self.datos['%D'].shift(1)))

    # Asigna un 1 cuando la orden es de compra y un -1 cuando es de venta, deja un 0 cuando hay que mantener.
    self.datos['Orden'] = 0
    self.datos.loc[condiciones_compra, 'Orden'] = 1 # Compra
    self.datos.loc[condiciones_venta, 'Orden'] = -1 # Venta
```

En esta función se establecen las condiciones de compra y venta:

- Las condiciones de compra se cumplen cuando %K es mayor que %D en el periodo actual y era menor o igual en el periodo anterior.
- Las condiciones de venta se cumplen cuando %K es menor que %D en el periodo actual y era mayor o igual en el periodo anterior.

Se crea una nueva columna en el dataframe para las órdenes de compra, inicializada a 0. Cuando se cumplen las condiciones de compra o venta, se establece a 1 o -1, respectivamente.

Volviendo a la clase Graficos, contamos con una función que realiza un gráfico de manera simple utilizando los precios de cierre y agrega los puntos de compra y venta de una forma muy visual y clara.

1 usage ▴ Iker Yañez *

```
def graficar_ordenes(self):
    st.header("Órdenes de compra y venta")
    fig = px.line(x=self.datos.index, y=self.datos['close'], labels={'x': 'Fecha', 'y': 'Precio de cierre'})

    compras = self.datos[self.datos['Orden'] == 1]
    ventas = self.datos[self.datos['Orden'] == -1]

    fig.add_trace(go.Scatter(x=compras.index, y=compras['close'], mode='markers', name='Compra',
                             marker=dict(color='green', size=10)))
    fig.add_trace(go.Scatter(x=ventas.index, y=ventas['close'], mode='markers', name='Venta',
                             marker=dict(color='red', size=10)))

    # Mostrar gráfico
    st.plotly_chart(fig)
```



Ordenes de compra y venta



2.3 Estructuración

Como se ha podido apreciar a lo largo de la memoria, el código se estructura en diferentes clases que contienen las diferentes funciones que el proyecto requiere. Estas clases se encuentran dentro del directorio src.

- Menu: Clase encargada de mostrar el menú y devolver la selección del par seleccionado por el usuario.
- DescargadorDatos: Clase encargada de la descarga del dataframe en función del par seleccionado.
- Indicadores: Clase encargada de realizar los cálculos estocásticos.
- Graficos: Clase encargada de mostrar los gráficos.

Finalmente, se integra todo el código en la ejecución principal, mostrada a continuación:



```
def main():
    try:
        st.title("Análisis Técnico de Criptomonedas")

        # Menú de selección
        menu = Menu()
        par = menu.menu()

        # Descarga de datos
        descargador = DescargadorDatos()
        datos = descargador.descargar_datos(par)

        if datos is not None:

            # Indicadores
            indicadores = Indicadores(datos)

            # Calcular indicadores
            indicadores.calcular_estocastico()
            indicadores.calcular_ordenes()

            # Graficos
            graficos = Graficos(datos)
            graficos.graficar_datos()
            graficos.graficar_estocastico()
            graficos.graficar_ordenes()

        else:
            st.warning("No se pudieron obtener los datos.")

    except Exception as e:
        st.error(f"Error en la ejecución principal: {e}")
```

Además, se ha realizado manejo de excepciones en la descarga de datos y ejecución principal (ver imágenes anteriores).

Adicionalmente, en el directorio `.github/workflows` encontramos el archivo necesario para la integración continua anteriormente mencionada. Gracias a esto, en cada subida de código se automatizará la reproducción del entorno y ejecución de los tests en máquinas Linux, Windows y macOS.



3. Ejecución

Para poder ejecutar el proyecto, se ha de abrir una terminal en la carpeta del proyecto y lanzar los siguientes comandos:

1. `pip install -r requirements.txt`
2. `streamlit run src/main.py`

De esta forma, se instalarán las librerías necesarias y se abrirá la aplicación web en el navegador.



4. Extra

4.1 Testing

Hemos considerado necesario hacer un test de integración simulando la ejecución del programa principal y a su vez realizando diversas comprobaciones.

```
def test_integration():
    with patch.object(Menu, attribute='menu', return_value="BTC/USD"):
        menu = Menu()
        par = menu.menu()
        assert par == "BTC/USD"

        descargador = DescargadorDatos()
        datos = descargador.descargar_datos(par)
        assert datos is not None

        indicadores = Indicadores(datos)
        indicadores.calcular_estocastico()
        indicadores.calcular_ordenes()

        verificar = ["%K", "%D", "%SD", "Orden"]
        for v in verificar:
            assert datos[v].notna().all()
```

Además, hemos realizado un test unitario de la clase Indicadores, debido a la importancia de los cálculos que realiza.

```
def test_calcular_estocastico():
    # Datos de prueba
    datos_prueba = pd.DataFrame({
        'open': [100, 110, 95, 105, 98, 105, 110, 120, 115, 105],
        'high': [120, 115, 100, 110, 102, 110, 115, 130, 120, 110],
        'low': [98, 105, 85, 95, 92, 95, 100, 110, 105, 95],
        'close': [105, 100, 90, 100, 95, 105, 105, 125, 110, 100]
    })

    # Crear instancia de Indicadores y calcular estocástico
    indicadores = Indicadores(datos_prueba)
    indicadores.calcular_estocastico(periodos=2)

    # Verificar que las columnas nuevas se hayan creado
    assert 'Min' in datos_prueba.columns
    assert 'Max' in datos_prueba.columns
    assert '%K' in datos_prueba.columns
    assert '%D' in datos_prueba.columns
    assert '%SD' in datos_prueba.columns

    # Verificar que los cálculos sean precisos
    assert datos_prueba['Min'].iloc[0] == 98
    assert datos_prueba['Max'].iloc[0] == 120
    assert datos_prueba['%K'].iloc[0] == 7/22*100
    assert datos_prueba['%D'].iloc[0] == datos_prueba['%K'].iloc[0:1].mean()
    assert datos_prueba['%SD'].iloc[0] == datos_prueba['%D'].iloc[0:1].mean()
```



Para lanzar los tests y ver tanto resultado como cobertura, se ha de lanzar el siguiente comando:

- `pytest --cov=.`

```
----- coverage: platform darwin, python 3.9.18-final-0 -----
```

Name	Stmts	Miss	Cover

src/DescargadorDatos.py	16	3	81%
src/Indicadores.py	15	0	100%
src/Menu.py	10	3	70%
src/test_indicadores.py	16	0	100%
src/test_integracion.py	18	0	100%

TOTAL	75	6	92%

Estos tests se ubican en el directorio src.

4.2 Reproducción del entorno

Como se ha mencionado anteriormente, el archivo requirements.txt contiene las librerías necesarias para la ejecución del programa, pudiendo instalarlas mediante pip.

4.3 Distribución

El proyecto ha sido distribuido como aplicación web mediante Streamlit. Se encuentra accesible desde el siguiente [enlace](#).