Jerson Alexander Peña Martínez.

PRÁCTICA OPEN DATA & VISUALIZACIÓN DINÁMICA

Requerimientos del sistema

- Versión de Python: 3.11.4 o superior
- Versión numpy: 1.21.5 o superior.
- Versión requests: 2.31.0 o superior.
- Versión Beautifulsoup4: 4.12.2 o superior.
- Versión Yfinance: 0.2.31 o superior.
- Versión pandas: 1.3.5 o superior.
- Versión plotly: 5.18.0 o superior.
- Version streamlit: 1.23.1 o superior.

Explicación del código

Carga de DataFrame con información de las compañías del SP500

Defino una función llamada load_data que utiliza la biblioteca pandas para crear un DataFrame con información procedente de Wikipedia.

```
def load_data():
    components = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0]
    return components.drop('CIK', axis=1).set_index('Symbol')
```

En este fragmento de código utilizo la biblioteca pandas para cargar datos desde una tabla en una página de Wikipedia. Utilizo la función read_html de pandas para extraer tablas HTML de una página web. En este caso, me quedo con la primera tabla de la página que contiene la lista de empresas del índice S&P 500.

Luego devuelvo este DataFrame, quitando una columna que no me interesa usando components.drop('CIK', axis=1). Y establezco como índice la columna con los tickers de las empresas usando set_index('Symbol'):.

Función de carga de dataframe para cotizaciones.

```
def load_quotes(ticker, period = '3y', interval = '1d'):
    #eliminar nas

    return yf.download(ticker, period= period, interval=interval)
```

Defino una función llamada load_quotes que utiliza la biblioteca yfinance para descargar datos históricos de precios de acciones.

Esta función recibe tres parámetros: ticker (símbolo de la acción), period (período de tiempo para el que se desean los datos, con un valor predeterminado de '3y' para 3 años), e interval (intervalo de tiempo entre los datos, con un valor predeterminado de '1d' para un día).

Luego para que sea dinámico, lo que hago es hacer un return yf.download(ticker, period=period, interval=interval), donde devuelvo un DataFrame con los datos históricos de precios de acciones. Los datos que componen el DataFrame son:

- Date: como fecha en la que se obtienen los datos.
- Open: precio de apertura.
- High: precio más alto alcanzado durante la sesión.
- Low: precio más bajo durante la sesión.
- Adj Close: precio de cierre ajustado por eventos coporativos.
- Volumen: cantidad total de acciones que se negocian durante la sesión.

Función scraper de Yahoofinance

```
#scraping key statistics from yahoofinance
def load_statics(ticker):
    key_statistics_dic = {}
    url= 'https://finance.yahoo.com/quote/{}/key-statistics?p={}'.format(ticker, ticker)
    key_statistics_dic[ticker] = {}

    headers = {'User-Agent': 'Chrome/117.0.5938.150'}
    page = requests.get(url, headers = headers)
    page_content = page.content
    soup = BeautifulSoup(page_content,'html.parser')
    tabl = soup.findAll("table", {'class': "W(100%) Bdcl(c)"})

    for t in tabl:
            rows = t.find_all("tr")
            for row in rows:
                if len(row.get_text(separator='|').split("|"))>0:
                    key_statistics_dic[ticker][row.get_text(separator='|').
                                split("|")[0]]=row.get_text(separator='|').split("|")[-1]

    key_statistics_dic[ticker] = pd.DataFrame(key_statistics_dic[ticker], index = [0]).T
        # Luego de crear el DataFrame key_statistics_dic[ticker], puedes aplicar fillna('')
    key_statistics_dic[ticker] = key_statistics_dic[ticker].fillna(' ')

    return key_statistics_dic[ticker]
```

Partes del código:

1. Creo un diccionario donde almacenaré los key_statistics por ticker, acontinuación tengo la url que voy a scrapear donde pongo las llaves donde se sustituirá por el ticker correspondiente en cada llamada.
2. Luego creo la parte de headers, la petición y almaceno la información en page para posteriormente pasarla por Beautifulsoup y obtener la información de las filas que me interesan. En este caso he cogido todas las tablas con la clase "W(100%) Bdcl(c)".

3. Después lo que hago es recorrer las tablas y las filas, y extraigo las estadísticas clave que todos comparten la misma clave que es “tr”.
    a. Recorro fila a fila. Como ví que tenían de separador “|” lo que hago es separarlas, y como aveces están vacias o no viene nada, lo que hago es asegurarme que sea mayor que 0.
    b. Despúes lo que hago es coger el primer elemento como clave (row.get_text(separator='|').split("|")[0])y el siguiente como valor (row.get_text(separator='|').split("|")[0]).

4. Creo un DataFrame de pandas con las estadísticas clave. Luego, transpongo el DataFrame y relleno los valores NaN con espacios en blanco.
5. Deveuelvo el dataframe con la información que necesito del ticker correspondiente.

Busqué varios indicadores que me parecían interesantes y los hice a mano para entenderlos. Realmente los busque de tradingview.

```
def ATR(DF, n=14):
    df = DF.copy()
    df['H-L'] = df['High'] - df['Low']
    df['H-PC'] = df['High'] - df['Adj Close'].shift(1)
    df['L-PC'] = df['High'] - df['Adj Close'].shift(1)
    df['TR'] = df[['H-L', 'H-PC', 'L-PC']].max(axis=1, skipna=False)
    df['ATR'] = df['TR'].ewm(com=n, min_periods=n).mean()
    return df['ATR']

def ADX(DF, n=20):
    df = DF.copy()
    df['ATR'] = ATR(df, n)
    df['upmove'] = df['High'] - df['High'].shift(1)
    df['downmove'] = df['Low'].shift(1) - df['Low']
    df['+dm'] = np.where((df['upmove'] > df['downmove']) & (df['upmove'] > 0), df['upmove'], 0)
    df['-dm'] = np.where((df['downmove'] > df['upmove']) & (df['downmove'] > 0), df['downmove'], 0)
    df['+di'] = 100 * (df['+dm'] / df['ATR']).ewm(span=n, min_periods=n).mean()
    df['-di'] = 100 * (df['-dm'] / df['ATR']).ewm(span=n, min_periods=n).mean()
    df['ADX'] = 100 * abs((df['+di'] - df['-di']) / (df['+di'] + df['-di'])).ewm(span=n, min_periods=n).mean()
    return df['ADX']

def Boll_Band(DF, n=14):
    df = DF.copy()
    df['MB'] = df['Adj Close'].rolling(n).mean()
    df['UB'] = df['MB'] + 2 * df['Adj Close'].rolling(n).std(ddof=0)
    df['LB'] = df['MB'] - 2 * df['Adj Close'].rolling(n).std(ddof=0)
    df['BB_Width'] = df["UB"] - df['LB']
    return df[['MB', 'UB', 'LB', 'BB_Width']]

def MACD(DF, a=12, b=26, c=9):
    df = DF.copy()
    df['ma_fast'] = df['Adj Close'].ewm(span=a, min_periods=a).mean()
    df['ma_slow'] = df['Adj Close'].ewm(span=b, min_periods=b).mean()
    df['macd'] = df['ma_fast'] - df['ma_slow']
    df['signal'] = df['macd'].ewm(span=c, min_periods=c).mean()
    return df.loc[:, ['macd', 'signal']]
```

Estos indicadores lo he cogido porque me parecían interesante hacer, las fórmulas las he sacado todas de tradingview. No me da tiempo a poder explicar esta parte así que he dejado los links donde se explican que significan. La selección que he dejado por defecto es una que he visto que recomiendan hacer.

ATR
https://www.tradingview.com/ideas/averagetruerange/?solution=43000501823

ADX
https://es.tradingview.com/scripts/directionalmovement/?solution=43000502250

Bollinger bands
https://www.tradingview.com/scripts/bollingerbands/?solution=43000501840

MACD
https://www.tradingview.com/scripts/bollingerbands/?solution=43000502344

Función

1. Defino la función donde voy a establecer todo lo necesario para la visualización.

```python
#1....
def main():
    st.title("Technical Analysis - Financial Indicators")

    st.sidebar.title("Options")
    show_companies_list = st.sidebar.checkbox('View companies list')

    if show_companies_list:
        components = load_data()
        st.subheader("Asset list")
        st.dataframe(components[['Security', 'GICS Sector', 'Date added', 'Founded']])

        st.sidebar.subheader('Select asset')
        asset = st.sidebar.selectbox('Click below to select a new asset',
                                     components.index.sort_values(), index=3,
                                     format_func=lambda x: x + ' - ' + components.loc[x].Security)
        st.title(components.loc[asset].Security)
        st.markdown(f"<h4 style='font-size: 14px; margin-top: -15px; margin-bottom: 5px;'>{components.loc[asset]['GICS Sub-Industry']}</h4>", unsafe_allow_html=True)

        # Información de la empresa sin el checkbox
        st.subheader("Información de la Empresa")
        st.table(components.loc[asset].iloc[1:])
```

- Lo primero que hago es elegir el título de la aplicación.
- Creo una barra lateral, con el título de "Options".
- Para empezar, muestro un dataframe con la información de las empresas del sp500.
- Creo una sección en la barra lateral para seleccionar un activo de una lista desplegable. La variable asset almacena el activo seleccionado.
- Muestro el nombre del activo seleccionado y su subindustria asociada. También se muestra una subsección con información de wikipedia de la empresa en forma de tabla.

2. Este fragmento de código tiene como objetivo identificar y mostrar los competidores de una empresa específica en la misma industria.

```python
#2......
# Todos los tickers en la misma industria ######
tickers_same_industry = components[components['GICS Sub-Industry'] == components.loc[asset]['GICS Sub-Industry']].index

# Creo un nuevo dataframe con la columna'asset' y 'Security'
asset_security_df = pd.DataFrame({'Asset': tickers_same_industry, 'Security': components.loc[tickers_same_industry, 'Security'].values})

# Display the table with asset names and their corresponding securities
st.subheader('Competidores de industria')
st.dataframe(asset_security_df)
```

- Creo una lista de símbolos (tickers_same_industry) que pertenecen a la misma subindustria que la empresa seleccionada (asset). Esto se logra filtrando el DataFrame components para encontrar las empresas con la misma subindustria.
- Creo un nuevo DataFrame (asset_security_df) que contiene dos columnas: 'Asset' (activo) y 'Security' (nombre de la empresa). Este DataFrame está formado por competidores en la misma industria.
- Uso Streamlit para mostrar una el dataframe anterior. Tiene como título Peers.

3. Este fragmento de código tiene como objetivo proporcionar una sección interactiva en la barra lateral de la aplicación. Esta sección permite al usuario ajustar el período y el intervalo de búsqueda de datos históricos de precios de acciones para el activo financiero seleccionado.

```
#3....
# Sección para modificar el período y el intervalo de la búsqueda
st.sidebar.subheader("Search Settings")
period_options = ["1d", "5d", "1mo", "3mo", "6mo", "1y", "2y", "5y", "10y", "ytd", "max"]
interval_options = ["1m", "2m", "5m", "15m", "30m", "60m", "90m", "1h", "1d", "5d", "1wk", "1mo", "3mo"]

selected_period = st.sidebar.selectbox('Select Period', period_options, index=5)
selected_interval = st.sidebar.selectbox('Select Interval', interval_options, index=3)

historic_data = load_quotes(asset, selected_period, selected_interval)
data = historic_data.copy().dropna()

# Agregar la sección
section = st.sidebar.slider('Number of quotes', min_value=30,
                            max_value=min([2000, data.shape[0]]),
                            value=500, step=10)
```

- Creo una subsección en la barra lateral con el título "Search Settings", indicando que esta parte del código se enfoca en ajustes de búsqueda.
- Defino las listas de opciones para el período y el intervalo de búsqueda.
- Se puede seleccionar el período y el intervalo deseados mediante cajas desplegables en la barra lateral. Los valores predeterminados son para un periodo de 5y y 1 día como intervalo, estos se pueden cambiar a gusto de quien lo use.
- Se cargan datos históricos de precios para el activo seleccionado, utilizando el período e intervalo seleccionados por el usuario. Se realiza una copia (data0.copy()) para preservar los datos originales y se eliminan los valores NaN.
- Creo un control deslizante (slider) en la barra lateral para poder seleccionar el número de cotizaciones a mostrar.

4. En este bloque de código, creo y configuro funciones dinámicas relacionadas con el cálculo de medias móviles para un conjunto de datos históricos de precios de acciones.

```python
#4.....
# Funciones Dinámicas - Medias Móviles
st.sidebar.subheader('Moving Averages')

# Me quedo con el precio de la acción.
data2 = data[-section:]['Adj Close'].to_frame('Adj Close')

# Media Móvil Simple (SMA)
sma_expander = st.sidebar.expander("Simple Moving Average (SMA)")
sma_enabled = sma_expander.checkbox('Enable SMA', value=True)
if sma_enabled:
    period = sma_expander.slider('SMA period', min_value=5, max_value=500, value=20, step=1)
    data[f'SMA {period}'] = data['Adj Close'].rolling(period).mean()
    data2[f'SMA {period}'] = data[f'SMA {period}'].reindex(data2.index)

# Media Móvil Simple 2 (SMA2)
sma2_expander = st.sidebar.expander("Simple Moving Average 2 (SMA2)")
sma2_enabled = sma2_expander.checkbox('Enable SMA2', value=True)
if sma2_enabled:
    period2 = sma2_expander.slider('SMA2 period', min_value=5, max_value=500, value=100, step=1)
    data[f'SMA2 {period2}'] = data['Adj Close'].rolling(period2).mean()
    data2[f'SMA2 {period2}'] = data[f'SMA2 {period2}'].reindex(data2.index)
```

- Creo una sección en la barra lateral titulada "Moving Averages" para incluir opciones relacionadas con medias móviles.
- Selecciono las últimas cotizaciones de precios de cierre ('Adj Close') según la selección realizada para el número de cotizaciones que se quieren mostrar (section). Esto se se hace para que la media se ajuste a lo que realmente se quiere ver.
- Creo un expandible (expander) en la barra lateral para la Media Móvil Simple (SMA), donde se puede habilitar o deshabilitar esta media móvil mediante un checkbox. Si se habilita, se puede configura el período de SMA, la cual se calcula y almacena en el DataFrame original (data) y en el DataFrame seleccionado (data2).
- Se hace lo mismo para SMA2.

5. En este bloque de código, se están configurando y visualizando las Bollinger Bands junto con los precios históricos de un activo financiero.

```python
#5....
# Bollinger Bands
boll_band_expander = st.sidebar.expander("Bollinger Bands")
enable_boll_band = boll_band_expander.checkbox('Enable Bollinger Bands', value=True)
n_boll_band = boll_band_expander.slider('Bollinger Bands period', min_value=5, max_value=50, value=20, step=1)
data[['MB', 'UB', 'LB', 'BB_Width']] = Boll_Band(data, n=n_boll_band)
data2[['MB', 'UB', 'LB', 'BB_Width']] = data[['MB', 'UB', 'LB', 'BB_Width']].reindex(data2.index)

# Gráfico
st.subheader('Chart')
# Convertir el indice a tipo Timestamp si es necesario
data2.index = pd.to_datetime(data2.index)

# gráficos con subplots
fig = make_subplots(rows=1, cols=1, shared_xaxes=True, subplot_titles=[f'{asset} - Historical Data'])

# Medias móviles (SMA y SMA2)
if sma_enabled:
    fig.add_trace(go.Scatter(x=data2.index, y=data2[f'SMA {period}'], mode='lines', name=f'SMA {period}', line=dict(width=1)))

if sma2_enabled:
    fig.add_trace(go.Scatter(x=data2.index, y=data2[f'SMA2 {period2}'], mode='lines', name=f'SMA2 {period2}', line=dict(width=1)))

# Añadir gráfico de Bollinger Bands
if enable_boll_band:
    fig.add_trace(go.Scatter(x=data2.index, y=data2['MB'], mode='lines', name='MB', line=dict(width=1, color='rgba(255, 0, 0, 0.5)')))
    fig.add_trace(go.Scatter(x=data2.index, y=data2['UB'], mode='lines', name='UB', line=dict(width=1, color='rgba(0, 0, 255, 0.5)')))
    fig.add_trace(go.Scatter(x=data2.index, y=data2['LB'], mode='lines', name='LB', line=dict(width=1, color='rgba(0, 0, 255, 0.5)')))
    fig.add_trace(go.Scatter(x=data2.index, y=data2['UB'], fill='tonexty', fillcolor='rgba(173, 216, 230, 0.3)', line=dict(width=0, color='rgba(0, 0, 255, 0.2)')))

# Gráfico de Precios
fig.add_trace(go.Scatter(x=data2.index, y=data2['Adj Close'], mode='lines', name='Adj Close', line=dict(width=1)))

# Configuración de ejes
fig.update_yaxes(title_text='Values', secondary_y=False)

# Leyenda debajo del gráfico

fig.update_layout(legend=dict(orientation='h', yanchor='bottom', y= 0, xanchor='auto', x=1))

# grid
fig.update_layout(
    xaxis=dict(showgrid=True, gridwidth=1, gridcolor='LightGray'),
    yaxis=dict(showgrid=True, gridwidth=1, gridcolor='LightGray'))

# gráfico
st.plotly_chart(fig)
```

- Creo una sección expandible en la barra lateral para configurar las Bollinger Bands, donde se puede habilitar o deshabilitar estas bandas mediante un checkbox. Se calculan las bandas y se actualizan en el DataFrame original (data) y en el DataFrame seleccionado (data2).
- Creo una sección de la interfaz gráfica titulada "Chart" para mostrar el gráfico. Se ajusta el índice de los datos seleccionados a tipo de objeto de fecha y hora. Se prepara la figura (fig) para el gráfico con subgráficos utilizando la función make_subplots.
- Añado líneas de medias móviles simples (SMA y SMA2) al gráfico si están habilitadas.
- He hecho uso de librerías como plotly.graph_objects para el grafico porque el que venía por defecto no se veía bien, también he usado make_subplots para poder poner una gráfico por encima de otro.
- Se añade un gráfico de las Bollinger Bands al gráfico principal si están habilitadas.
- Por último, hago uso de streamlit para mostrar el gráfico.

6. En este bloque de código, se configuran y visualizan dos indicadores técnicos, el Average True Range (ATR) y el Moving Average Convergence Divergence (MACD).

```
#6....
# Funciones Dinámicas - Technical Indicators

st.sidebar.subheader('Technical Indicators')

# ADX
adx_expander = st.sidebar.expander("Average True Range (ATR)")
enable_adx = adx_expander.checkbox('Enable ADX', value=True)
n_adx = adx_expander.slider('ADX period', min_value=5, max_value=50, value=20, step=1)
data['ADX'] = ADX(data, n=n_adx)
data2['ADX'] = data['ADX'].reindex(data2.index)



# MACD
macd_expander = st.sidebar.expander("Moving Average Convergence Divergence (MACD)")
enable_macd = macd_expander.checkbox('Enable MACD', value=True)
a_macd = macd_expander.slider('MACD fast period', min_value=5, max_value=50, value=12, step=1)
b_macd = macd_expander.slider('MACD slow period', min_value=5, max_value=50, value=26, step=1)
c_macd = macd_expander.slider('MACD signal period', min_value=5, max_value=50, value=9, step=1)
data[['MACD', 'SIGNAL']] = MACD(data, a=a_macd, b=b_macd, c=c_macd)
data2[['MACD', 'SIGNAL']] = data[['MACD', 'SIGNAL']].reindex(data2.index)

# Gráfico ADX
if enable_adx:
    fig_adx = go.Figure()
    fig_adx.add_trace(go.Scatter(x=data2.index, y=data2['ADX'], mode='lines', name='ADX'))
    fig_adx.update_layout(title='Average True Range (ADX)',
                          xaxis_title='Date', yaxis_title='ADX',
                          showlegend=True)
    st.plotly_chart(fig_adx)


# Gráfico MACD
if enable_macd:
    fig_macd = go.Figure()
    fig_macd.add_trace(go.Scatter(x=data2.index, y=data2['MACD'], mode='lines', name='MACD'))
    fig_macd.add_trace(go.Scatter(x=data2.index, y=data2['SIGNAL'], mode='lines', name='Signal'))
    fig_macd.update_layout(title='Moving Average Convergence Divergence (MACD)',
                           xaxis_title='Date', yaxis_title='Values',
                           showlegend=True)
    st.plotly_chart(fig_macd)
```

Average True Range (ATR):

- Se crea sección en la barra lateral para configurar y habilitar el ATR.
- Se puede ajustar el período del ATR mediante un control deslizante.
- Se calcula el ATR para el conjunto de datos y se agrega a la columna 'ADX'.
- Se muestra un gráfico interactivo del ATR si está habilitado.

Moving Average Convergence Divergence (MACD):

- Se crea una sección en la barra lateral para configurar y habilitar el MACD.
- Se puede puede ajustar los períodos rápidos, lentos y de señal mediante controles deslizantes.
- Se calcula el MACD y la señal para el conjunto de datos y se agregan a las columnas 'MACD' y 'SIGNAL'.
- Se muestra un gráfico interactivo del MACD y la señal si está habilitado.

7. En esta parte del código realiza un análisis estadístico de precios para las acciones de la industria de la acción seleccionada. Y por último se devuelve el los key statatistic de yahoofinance.

```
##7..|.
# Estadísticas de precios para la industria
st.sidebar.subheader('Statistics for Industry')

# Botón de selección para tipo de dato (Adj Close o Volumen)
data_type = st.sidebar.radio("Select Data Type", ['Adj Close', 'Volumen'], index=0)

# Crear un DataFrame con los datos seleccionados para todas las acciones en la misma industria
if data_type == 'Adj Close':
    data_industry = yf.download(tickers_same_industry.tolist(), period=selected_period, interval=selected_interval)['Adj Close']
else:
    data_industry = yf.download(tickers_same_industry.tolist(), period=selected_period, interval=selected_interval)['Volume']


   # Calcular las estadísticas descriptivas
industry_stats = data_industry.describe().round(2)

# Mostrar la tabla con el nombre de la columna como el ticker y el título como estadísticas del tipo de dato seleccionado
st.subheader(f'Industry {data_type} Statistics')
st.table(industry_stats.T.style.format("{:n}").set_caption(f'Industry {data_type} Statistics'))

# Main code
key_statistics_data = {}

# Load key statistics data for each ticker
for ticker in tickers_same_industry:
    key_statistics_data[ticker] = load_statics(ticker)

# Display key statistics in a single table
st.subheader('Key Statistics')

# Create an empty dataframe to store the concatenated data
concatenated_data = pd.DataFrame()

# Concatenate the key statistics data into a single dataframe
for ticker in tickers_same_industry:
    # Add the data for each ticker to the concatenated dataframe
    concatenated_data[ticker] = key_statistics_data[ticker].squeeze()

# Display the concatenated dataframe as a single table
st.write(concatenated_data)
```

- Creo un encabezado en la barra lateral "Statistics for Industry".
- Agrego dos botones de selección en la barra lateral llamado "Select Data Type", que permite: 'Adj Close' y 'Volumen'. La opción seleccionada se almacena en la variable data_type.
- Descargo los datos usando yfinance, creo un DataFrame llamado data_industry que contiene los datos seleccionados para todas las acciones en la misma industria. El tipo de datos descargados depende de la selección la selección que hagamos.
- Calculo las estadísticas descriptivas (como media, mediana, etc.) para los datos descargados de la industria. Los resultados se almacenan en el DataFrame industry_stats.
- Por último, muestro una tabla que representa las estadísticas de la industria. Las columnas de la tabla tienen los nombres de los tickers, y las filas contienen las estadísticas descriptivas.

En el aparatado de Main code.

- Creo un diccionario vacío llamado key_statistics_data para almacenar las estadísticas clave de cada acción en la industria.
- Uso un bucle for para recorrer cada ticker en tickers_same_industry. Para cada ticker, llamo a la función load_statics(ticker) para cargar las estadísticas clave asociadas con esa acción. Estas estadísticas clave se almacenan en el diccionario key_statistics_data con el ticker como clave.
- Creo una subcabecera con el texto 'Key Statistics'
- Crea un DataFrame vacío llamado concatenated_data donde almacenaré la informaciónde cada ticker con sus key statistics.
- Hago uso de un bucle for para recorrer cada ticker en tickers_same_industry.
- Luego para cada ticker, tomo las key statistics del diccionario key_statistics_data y las agrego al DataFrame concatenated_data en una columna correspondiente al ticker.
- La función squeeze se utiliza para convertir el DataFrame de key statistics en una Serie si es necesario.
- Por último se muestra el DataFrame haciendo uso de st.write.