import streamlit as st

import pandas as pd

import io

def load\_sheet\_data(uploaded\_file, sheet\_name):

    data = pd.read\_excel(uploaded\_file, sheet\_name=sheet\_name)

    return data

def setup\_streamlit\_app():

    st.set\_page\_config(page\_title="Investor Dashboard", page\_icon="📊", layout="wide")

    st.markdown("<h1 style='text-align: center;'>Data 📊</h1>", unsafe\_allow\_html=True)

    # Gestion du fichier téléchargé

    if 'uploaded\_file' in st.session\_state:

        uploaded\_file = st.session\_state['uploaded\_file']

        xl = pd.ExcelFile(uploaded\_file)

        sheet\_names = xl.sheet\_names

        # Chargement des noms de colonnes de la feuille "MAX" pour utilisation ultérieure

        if "MAX" in sheet\_names:

            max\_data = pd.read\_excel(uploaded\_file, sheet\_name="MAX")

            st.session\_state['max\_columns'] = max\_data.columns.tolist()  # Stockage des noms des colonnes

        selected\_sheet = st.selectbox('Visualisation des données', sheet\_names, index=sheet\_names.index('MAX') if 'MAX' in sheet\_names else 0)

        data = load\_sheet\_data(uploaded\_file, selected\_sheet)

        st.session\_state['df'] = data

        st.session\_state['selected\_sheet\_index'] = sheet\_names.index(selected\_sheet)

        st.dataframe(data)

    else:

        uploaded\_file = st.file\_uploader("Importer les données (.xlsm)", type=['xlsm'])

        if uploaded\_file is not None:

            st.session\_state['uploaded\_file'] = io.BytesIO(uploaded\_file.getvalue())

            st.experimental\_rerun()

def main():

    setup\_streamlit\_app()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Tableau   
import streamlit as st

import pandas as pd

import plotly.express as px

from ta.trend import SMAIndicator, EMAIndicator

from ta.momentum import RSIIndicator, StochasticOscillator

import plotly.graph\_objects as go

from ta.volatility import BollingerBands

from ta.trend import MACD

from ta.momentum import ROCIndicator, StochRSIIndicator

from ta.trend import CCIIndicator

st.set\_page\_config(page\_title="Investor Dashboard", page\_icon="💰", layout="wide")

st.markdown("""

<style>

    /\* Supprimer la marge supérieure par défaut \*/

    .reportview-container .main .block-container {

        padding-top: 0rem; /\* Réduire cette valeur selon vos besoins, 0rem pour aucune marge \*/

    }

    /\* Ajuster l'espacement autour du corps du rapport \*/

    .reportview-container {

        padding: 0rem 1rem;  /\* Ajuster la marge horizontale ici \*/

    }

</style>

""", unsafe\_allow\_html=True)

st.markdown("<h1 style='text-align: center;'>💰Tableau de Bord de l'investisseur 💰</h1>", unsafe\_allow\_html=True)

# Zone de sélection du marché

st.markdown("#### Sélectionnez le marché")

market = st.selectbox("Marché:", ["BRVM"], index=0, help="Sélectionnez le marché pour afficher les données correspondantes.")

# Section de sélection de l'action

st.markdown("#### Sélectionnez l'action à analyser")

if 'max\_columns' in st.session\_state:

        # Exclure la colonne 'Date' des options

        options = [col for col in st.session\_state['max\_columns'] if col != 'Date']

        action = st.selectbox("Action :", options,help="Sélectionnez le marché pour afficher les données correspondantes")

        #st.write(f"Action sélectionnée : {action}")

else:

        st.error("Les noms des colonnes n'ont pas été chargés. Veuillez d'abord charger le fichier sur la page de données.")

# Définition des proportions pour les colonnes : les deux premières colonnes sont plus larges que la troisième

col1, col2, col3 = st.columns([3, 3, 1])

# Ajout d'un contenu dans la première colonne : Analyse Technique

with col1:

    st.markdown("#### 📈 Analyse Technique")

    if 'max\_columns' in st.session\_state and action:

        # Chargement et préparation des données

        data = pd.read\_excel(st.session\_state['uploaded\_file'], sheet\_name='COURS')

        data = data[::-1]  # Inverser si nécessaire

        data['Date'] = pd.to\_datetime(data['Date'])

        filtered\_data = data[data['Date'].dt.year >= 2018]

        volume\_data = pd.read\_excel(st.session\_state['uploaded\_file'], sheet\_name='VOLUME')

        volume\_data['Date'] = pd.to\_datetime(volume\_data['Date'])

        filtered\_volume\_data = volume\_data[volume\_data['Date'].dt.year >= 2018]

       # Utilisation des onglets pour différents indicateurs

    tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8,tab9 = st.tabs(["Cours", "Volume", "RSI", "MACD", "Moyennes Mobiles", "Bandes de Bollinger", "EMA", "ROC","Momentum"])

    with tab1:

        # Tracer le graphique de l'action sélectionnée

        fig = px.line(filtered\_data, x='Date', y=action, height=400)

        st.plotly\_chart(fig, use\_container\_width=True)

    with tab2:

        # Tracer le graphique du volume

        volume\_data = pd.read\_excel(st.session\_state['uploaded\_file'], sheet\_name='VOLUME')

        volume\_data['Date'] = pd.to\_datetime(volume\_data['Date'])

        filtered\_volume\_data = volume\_data[volume\_data['Date'].dt.year >= 2018]

        fig = px.bar(filtered\_volume\_data, x='Date', y=action, color\_discrete\_sequence=['red'], labels={action: "Volume"}, height=400)

        st.plotly\_chart(fig, use\_container\_width=True)

    with tab3:

        # Calculer le RSI pour la période de 14 jours

        rsi = RSIIndicator(filtered\_data[action], window=14).rsi()

        # Tracer le RSI sur un nouveau graphique

        fig\_rsi = px.line(filtered\_data, x='Date', y=rsi)

        fig\_rsi.update\_layout(

            xaxis\_title='Date',

            yaxis\_title='RSI ',

            yaxis=dict(

                range=[0, 100],  # Typique pour RSI pour voir les niveaux de surachat (>70) et de survente (<30)

                title='RSI (14)' ),

            template='plotly\_white',height=400

        )

        st.plotly\_chart(fig\_rsi, use\_container\_width=True)

    with tab4:

        # Calculer le MACD

        macd\_indicator = MACD(close=filtered\_data[action], window\_slow=26, window\_fast=12, window\_sign=9)

        macd\_line = macd\_indicator.macd()

        signal\_line = macd\_indicator.macd\_signal()

        histogram = macd\_indicator.macd\_diff()

        # Créer un nouveau graphique pour le MACD

        fig = go.Figure()

        # Ajouter la ligne MACD

        fig.add\_trace(go.Scatter(x=filtered\_data['Date'], y=macd\_line, mode='lines', name='MACD Line', line=dict(color='blue')))

        # Ajouter la ligne de signal

        fig.add\_trace(go.Scatter(x=filtered\_data['Date'], y=signal\_line, mode='lines', name='Signal Line', line=dict(color='red')))

        # Ajouter l'histogramme avec des couleurs conditionnelles

        colors = ['red' if val < 0 else 'green' for val in histogram]

        fig.add\_trace(go.Bar(x=filtered\_data['Date'], y=histogram, name='Histogram', marker\_color=colors))

        # Configurer la légende pour qu'elle soit en haut et horizontale

        fig.update\_layout(

            legend=dict(

                orientation='h',

                yanchor='bottom',

                y=1.02,  # Légèrement au-dessus du haut du graphique

                xanchor='right',

                x=1

            ),

            xaxis\_title='Date',

            yaxis\_title='MACD and Signal',

            yaxis=dict(

                title='MACD and Signal'

            ),

            yaxis2=dict(

                title='Histogram',

                overlaying='y',

                side='right',

                showgrid=False,  # Désactiver la grille pour l'axe secondaire

            ),

            template='plotly\_white',height=400

        )

        # Afficher le graphique dans Streamlit

        st.plotly\_chart(fig, use\_container\_width=True)

    with tab5:

        # Calculer et tracer les Moyennes Mobiles

        fig = px.line(filtered\_data, x='Date', y=action)

        # Calcul et affichage des moyennes mobiles pour 7, 21 et 100 jours

        sma\_7 = SMAIndicator(filtered\_data[action], window=7).sma\_indicator()

        sma\_21 = SMAIndicator(filtered\_data[action], window=21).sma\_indicator()

        sma\_100 = SMAIndicator(filtered\_data[action], window=100).sma\_indicator()

        # Ajout des lignes de moyennes mobiles sur le graphique des cours

        fig.add\_scatter(x=filtered\_data['Date'], y=sma\_7, mode='lines', name='SMA 7', line=dict(color='green', width=1))

        fig.add\_scatter(x=filtered\_data['Date'], y=sma\_21, mode='lines', name='SMA 21', line=dict(color='orange', width=1))

        fig.add\_scatter(x=filtered\_data['Date'], y=sma\_100, mode='lines', name='SMA 100', line=dict(color='red', width=1))

        # Mise à jour du layout pour inclure une légende horizontale en haut

        fig.update\_layout(

            legend=dict(

                title='Indicateurs',

                orientation='h',

                yanchor="bottom",

                y=1.02,

                xanchor="right",

                x=1

            ),

            xaxis\_title='Date',

            yaxis\_title=f'Prix de {action}',

            template='plotly\_white'

        )

        # Afficher le graphique final avec les moyennes mobiles si sélectionnées

        st.plotly\_chart(fig, use\_container\_width=True)

    with tab6:

        # Calculer les Bandes de Bollinger pour la fenêtre de 20 jours et 2 écarts-types

        bb\_indicator = BollingerBands(close=filtered\_data[action], window=20, window\_dev=2)

        bb\_high\_band = bb\_indicator.bollinger\_hband()

        bb\_low\_band = bb\_indicator.bollinger\_lband()

        bb\_mid\_band = bb\_indicator.bollinger\_mavg()

        # Tracer le cours de l'action avec les Bandes de Bollinger

        fig\_bb = go.Figure()

        # Ajouter la bande supérieure

        fig\_bb.add\_trace(go.Scatter(x=filtered\_data['Date'], y=bb\_high\_band, name='Bande Supérieure', line=dict(color='red', width=1)))

        # Ajouter la bande inférieure

        fig\_bb.add\_trace(go.Scatter(x=filtered\_data['Date'], y=bb\_low\_band, name='Bande Inférieure', line=dict(color='orange', width=1)))

        # Ajouter la bande moyenne

        fig\_bb.add\_trace(go.Scatter(x=filtered\_data['Date'], y=bb\_mid\_band, name='Bande Moyenne', line=dict(color='green', width=2)))

        # Ajouter le cours de l'action

        fig\_bb.add\_trace(go.Scatter(x=filtered\_data['Date'], y=filtered\_data[action], name='Cours de l\'action', line=dict(color='blue', width=2)))

        # Mise à jour de la mise en page du graphique

        fig\_bb.update\_layout(

            xaxis\_title='Date',

            yaxis\_title=f'Prix de {action}',

            template='plotly\_white',

            height=400,

            legend=dict(

                orientation='h',

                yanchor='bottom',

                y=1.02,

                xanchor='right',

                x=1

            )

        )

        st.plotly\_chart(fig\_bb, use\_container\_width=True)

    with tab7:

        # Calculer et tracer les Moyennes Mobiles Expo

        fig = px.line(filtered\_data, x='Date', y=action)

        # Calcul et affichage des moyennes mobiles pour 14 et 100 jours

        ema\_14 = EMAIndicator(filtered\_data[action], window=7).ema\_indicator()

        ema\_100 = EMAIndicator(filtered\_data[action], window=100).ema\_indicator()

        # Ajout des lignes de moyennes mobiles sur le graphique des cours

        fig.add\_scatter(x=filtered\_data['Date'], y=ema\_14, mode='lines', name='EMA 7', line=dict(color='red', width=1.5))

        fig.add\_scatter(x=filtered\_data['Date'], y=ema\_100, mode='lines', name='EMA 100', line=dict(color='green', width=1))

        # Mise à jour du layout pour inclure une légende horizontale en haut

        fig.update\_layout(

            legend=dict(

                title='Indicateurs',

                orientation='h',

                yanchor="bottom",

                y=1.02,

                xanchor="right",

                x=1

            ),

            xaxis\_title='Date',

            yaxis\_title=f'Prix de {action}',

            template='plotly\_white'

        )

        # Afficher le graphique final avec les moyennes mobiles si sélectionnées

        st.plotly\_chart(fig, use\_container\_width=True)

    with tab8:

        # Calculer le ROC pour une période de 12 jours

        roc\_indicator = ROCIndicator(close=filtered\_data[action], window=12)

        roc = roc\_indicator.roc()

        # Tracer le ROC sur un nouveau graphique

        fig\_roc = px.line(filtered\_data, x='Date', y=roc)

        fig\_roc.update\_layout(

            xaxis\_title='Date',

            yaxis\_title='ROC',

            template='plotly\_white',

            height=400

        )

        st.plotly\_chart(fig\_roc, use\_container\_width=True)

    with tab9:

        # Calculer le Momentum pour une période de 10 jours

        momentum\_indicator = SMAIndicator(close=filtered\_data[action], window=10)  # Utilisation de SMA comme proxy pour le momentum

        momentum = momentum\_indicator.sma\_indicator()

        # Tracer le Momentum sur un nouveau graphique

        fig\_momentum = px.line(filtered\_data, x='Date', y=momentum)

        fig\_momentum.update\_layout(

            xaxis\_title='Date',

            yaxis\_title='Momentum',

            template='plotly\_white',

            height=400

        )

        st.plotly\_chart(fig\_momentum, use\_container\_width=True)

    # Création d'une liste pour contenir les signaux

    signals = []

    # Moyennes Mobiles

    sma\_7 = SMAIndicator(filtered\_data[action], window=7).sma\_indicator()

    sma\_21 = SMAIndicator(filtered\_data[action], window=21).sma\_indicator()

    sma\_100 = SMAIndicator(filtered\_data[action], window=100).sma\_indicator()

    last\_price = filtered\_data[action].iloc[-1]

    # Signaux pour les moyennes mobiles

    signals.extend([

            {

                'Nom': 'Moyenne Mobile (7)',

                'Valeur': round(sma\_7.iloc[-1],2),

                'Action': 'Acheter' if sma\_7.iloc[-1] < last\_price else 'Vendre' if sma\_7.iloc[-1] > last\_price else 'Neutre'

            },

            {

                'Nom': 'Moyenne Mobile (21)',

                'Valeur': round(sma\_21.iloc[-1],2),

                'Action': 'Acheter' if sma\_21.iloc[-1] < last\_price else 'Vendre' if sma\_21.iloc[-1] > last\_price else 'Neutre'

            },

            {

                'Nom': 'Moyenne Mobile (100)',

                'Valeur': round(sma\_100.iloc[-1],2),

                'Action': 'Acheter' if sma\_100.iloc[-1] < last\_price else 'Vendre' if sma\_100.iloc[-1] > last\_price else 'Neutre'

            }

        ])

    # RSI

    rsi = RSIIndicator(filtered\_data[action], window=14).rsi()

    last\_rsi = round(rsi.iloc[-1], 2)

    signals.append({

            'Nom': 'Indice de force relative (14)',

            'Valeur': last\_rsi,

            'Action': 'Acheter' if last\_rsi < 30 else 'Vendre' if last\_rsi > 70 else 'Neutre'

        })

    # MACD

    macd\_indicator = MACD(close=filtered\_data[action], window\_slow=26, window\_fast=12, window\_sign=9)

    macd\_diff = round(macd\_indicator.macd\_diff().iloc[-1],2)

    signals.append({

            'Nom': 'MACD (12,26)',

            'Valeur': macd\_diff,

            'Action': 'Acheter' if macd\_diff > 0 else 'Vendre' if macd\_diff < 0 else 'Neutre'

        })

    # Ajout du signal de Bandes de Bollinger

    indicator\_bb = BollingerBands(close=filtered\_data[action], window=20, window\_dev=2)

    bb\_upper = indicator\_bb.bollinger\_hband\_indicator().iloc[-1]

    bb\_lower = indicator\_bb.bollinger\_lband\_indicator().iloc[-1]

    # La logique de signal pour les Bandes de Bollinger peut être basée sur le croisement des prix avec les bandes

    if filtered\_data[action].iloc[-1] > bb\_upper:

            bb\_action = 'Vendre'  # Prix au-dessus de la bande supérieure

    elif filtered\_data[action].iloc[-1] < bb\_lower:

            bb\_action = 'Acheter'  # Prix en dessous de la bande inférieure

    else:

            bb\_action = 'Neutre'

    signals.append({

            'Nom': 'Bandes de Bollinger (20, 2)',

            'Valeur': f"Upper: {bb\_upper}, Lower: {bb\_lower}",

            'Action': bb\_action

        })

        # EMA

    ema\_indicator = EMAIndicator(close=filtered\_data[action], window=14)

    ema = ema\_indicator.ema\_indicator()

    last\_ema = round(ema.iloc[-1], 2)

    signals.append({

            'Nom': 'EMA (14)',

            'Valeur': last\_ema,

            'Action': 'Acheter' if last\_price > last\_ema else 'Vendre' if last\_price < last\_ema else 'Neutre'

        })

    ema\_indicator = EMAIndicator(close=filtered\_data[action], window=100)

    ema = ema\_indicator.ema\_indicator()

    last\_ema = round(ema.iloc[-1], 2)

    signals.append({

            'Nom': 'EMA (100)',

            'Valeur': last\_ema,

            'Action': 'Acheter' if last\_price > last\_ema else 'Vendre' if last\_price < last\_ema else 'Neutre'

        })

    # Momentum

    momentum\_indicator = SMAIndicator(filtered\_data[action], window=10).sma\_indicator()

    last\_momentum = round(momentum\_indicator.iloc[-1], 2)

    signals.append({

            'Nom': 'Momentum (10)',

            'Valeur': last\_momentum,

            'Action': 'Acheter' if last\_momentum > 0 else 'Vendre' if last\_momentum < 0 else 'Neutre'

        })

    # Rate of Change (ROC)

    roc\_indicator = ROCIndicator(filtered\_data[action], window=12)

    last\_roc = round(roc\_indicator.roc().iloc[-1], 2)

    signals.append({

            'Nom': 'ROC (12)',

            'Valeur': last\_roc,

            'Action': 'Acheter' if last\_roc > 0 else 'Vendre' if last\_roc < 0 else 'Neutre'

        })

    def display\_signals(signals):

            # En-têtes des colonnes

            col1, col2, col3 = st.columns([2, 2, 2])

            with col1:

                st.markdown("\*\*Nom\*\*")

            with col2:

                st.markdown("\*\*Valeur\*\*")

            with col3:

                st.markdown("\*\*Action\*\*")

            # Données des signaux

            for signal in signals:

                col1, col2, col3 = st.columns([3, 3, 3])

                with col1:

                    st.write(signal['Nom'])

                with col2:

                    st.write(f"{signal['Valeur']}")

                # Utilisation du HTML pour colorer les actions en rouge ou vert

                action\_color = "red" if signal['Action'] == "Vendre" else "blue" if signal['Action'] == "Acheter" else "black"

                with col3:

                    st.markdown(f"<span style='color: {action\_color};'>{signal['Action']}</span>", unsafe\_allow\_html=True)

    # Affichage conditionnel des signaux

    if 'show\_signals' in st.session\_state and st.session\_state['show\_signals']:

            display\_signals(signals)

with col2:

        st.markdown("#### 📊 Analyse Fondamentale")

        if 'uploaded\_file' in st.session\_state and action:

            indices\_data = pd.read\_excel(st.session\_state['uploaded\_file'], sheet\_name='INDICES')

            if action in indices\_data.columns:

                indices\_list = indices\_data[action].dropna().tolist()

                if indices\_list:

                    selected\_index = st.selectbox('', indices\_list)

                else:

                    st.write(f"Aucun indice trouvé pour l'action {action}.")

            else:

                st.write(f"L'action {action} n'est pas trouvée dans la feuille 'INDICES'.")

            if 'selected\_index' in locals() or 'selected\_index' in globals():

                cours\_data = pd.read\_excel(st.session\_state['uploaded\_file'], sheet\_name='COURS')

                cours\_data['Date'] = pd.to\_datetime(cours\_data['Date'])

                if selected\_index in cours\_data.columns:

                    valid\_dates = cours\_data[cours\_data[selected\_index].notna()]['Date']

                    if not valid\_dates.empty:

                        date\_debut = valid\_dates.min()

                        filtered\_data = cours\_data[cours\_data['Date'] >= date\_debut]

                        indice\_cours = filtered\_data[selected\_index]

                        df\_to\_plot = pd.DataFrame({

                            'Date': filtered\_data['Date'],

                            selected\_index: indice\_cours

                        })

                        fig = px.line(df\_to\_plot, x='Date', y=selected\_index, title=f'Cours de l\'indice {selected\_index}')

                        st.plotly\_chart(fig)

                    else:

                        st.error(f"Aucune donnée valide trouvée pour l'indice {selected\_index}.")

                else:

                    st.error(f"L'indice {selected\_index} n'est pas trouvé dans la feuille 'COURS'.")

        st.markdown('</div>', unsafe\_allow\_html=True)

# Ajout d'un contenu dans la troisième colonne : Vue Globale

with col3:

    st.markdown("#### 🌍 Vue Globale")

    st.write("Vue globale ici...")

# Ajout du texte défilant en rouge en bas de la page

st.markdown("""

<footer style="position: fixed; width: 100%; bottom: 0; background-color: white; text-align: center;">

   <marquee style="color: red; font-size: 16px;">

        ⚠️ Attention : Les informations présentées ici résultent d'une analyse personnelle et sont fournies à titre informatif uniquement. Elles ne doivent pas être interprétées comme des conseils d'investissement.⚠️

    </marquee>

</footer>

""", unsafe\_allow\_html=True)