Python para Bovespa

Utilização das bibliotecas Fundamentus e Yahoo Finance para análise de ativos da Bolsa Brasileira

Silas Valério

Índice

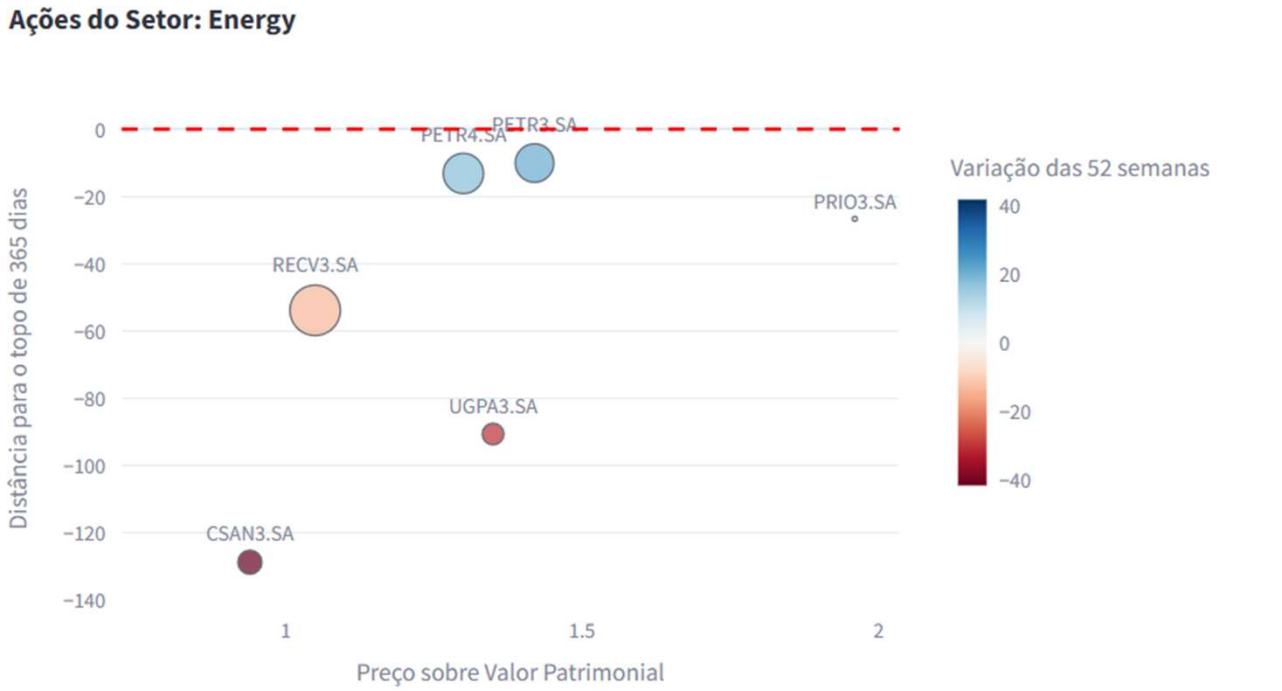
Introdução

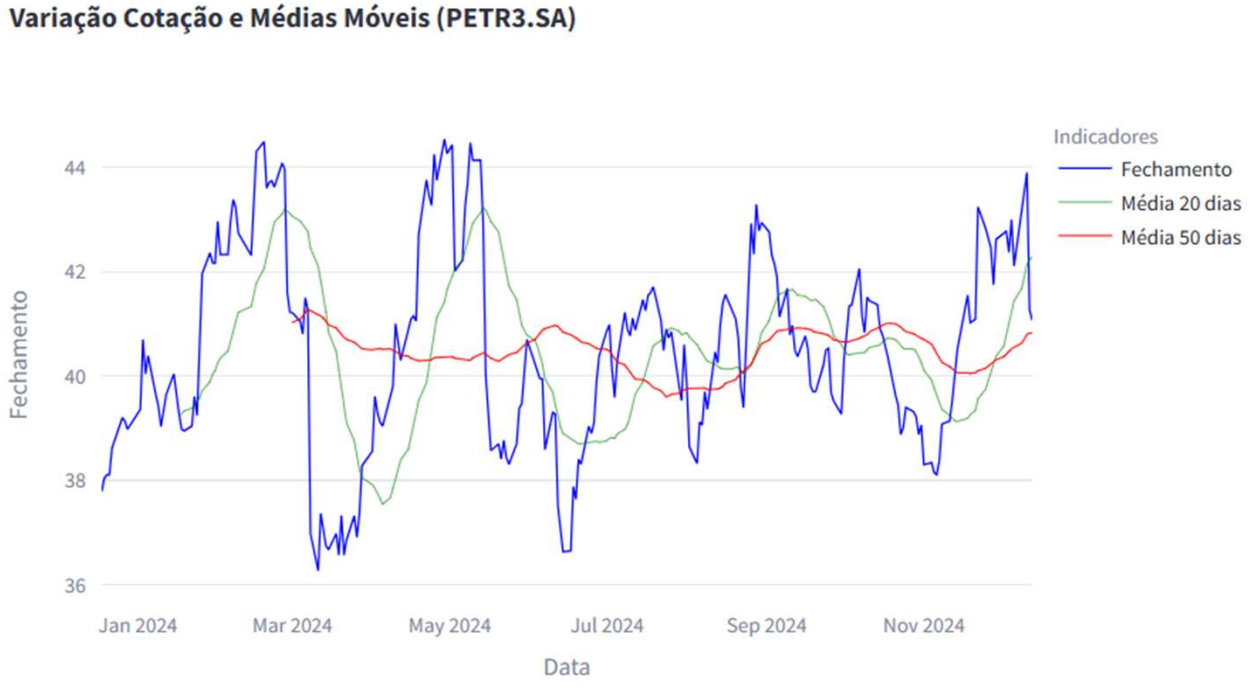
Este e-book descreve os passos necessários para criar um projeto que utiliza Python e Streamlit para gerar gráficos interativos para a análise de ações negociadas na Bovespa.

O objetivo principal é demonstrar como construir gráficos utilizando dados fundamentalistas extraídos com a biblioteca Yahoo Finance.

Aspecto Final

Ao final, os gráficos deverão ter esse aspecto:





1 – Preparação

Instalação de Bibliotecas:

As principais bibliotecas utilizadas foram:

- streamlit: para a interface interativa.
- plotly: para gráficos interativos.
- yfinance: para obter dados de ações.
- pandas e numpy: para manipulação de dados.

Comando para instalação das bibliotecas:

```
!pip install git+https://github.com/mv/fundamentus-api
!pip install streamlit plotly yfinance
```

1.1 – Importação das bibliotecas

Após a instalação, as bibliotecas devem ser importadas.

Além das instaladas na seção anterior, será necessário importar a biblioteca DATETIME para trabalhar com datas.

```
import fundamentus
import pandas as pd
import yfinance as yf
from datetime import datetime, timedelta
```

2 – Carregando Dados Fundamentalistas

Com a utilização da biblioteca Fundamentus, são carregados em *dataframe* os dados das empresas com ações negociadas na Bovespa.

```
df = fundamentus.get_resultado()
```

Para facilitar a compreensão e a manipulação da *dataframe*, as *strings* com o nome das colunas recebidas são carregados em variáveis com nomes amigáveis.

```
cotacao='cotacao'
patrimonioLiquido='pl'
precoPorValorPatrimonial='pvp'
precoPorReceitaLiquida='psr'
dividendYield='dy'
precoPorAtivosTotais='pa'
precoPorCapitalGiro='pcg'
precoPorEbit='pebit'
precoPorAtivoCirculante='pacl'
valorEmpresaPorEbit='evebit'
valorEmpresaPorEbitda='evebitda'
ebitPorReceitaLiquida='mrgebit'
lucroLiquidoPorReceitaLiquida='mrgliq'
lucroPorAcao='lucroPorAcao'
retornoSobreCapitalInvestido='roic'
retornoSobrePatrimonioLiquido='roe'
liquidezCorrente='liqc'
volumeNegociacao2Meses='liq2m'
patrimonioLiquido='patrliq'
dividaBrutaSobrePatrimonio='divbpatr'
crescimentoReceita5Anos='c5y'
tickerYfinance='tickeryf'
```

3 – Filtrando Empresas

Nessa etapa, o *dataframe* será filtrado buscando eliminar empresas que não possuam boa performance.

Cada usuário deverá adequar o filtro aos seus interesses e propósitos.

```
def filtrar_acoes(df):
    # Critérios de seleção
     filtro = (
        (df['patrimonioLiquido'] > 0) &
        (df['lucroLiquidoPorReceitaLiquida'] > 0) &
        (df['dividendYield'] > 0.05) &
        (df['valorEmpresaPorEbit'] ⩾ 1) &
        (df['retornoSobrePatrimonioLiquido'] > 0.2) &
        (df['dividaBrutaSobrePatrimonio'] ⩽ 0.5)  # Ajuste no critério de dívida
    )
    return df[filtro]
```

A função retorna um filtro que deverá ser aplicado para criar um novo *dataframe* por opções de conveniência.

```
acoes_filtradas = filtrar_acoes(df)
```

4 – Integrando com Yahoo Finance

A biblioteca Fundamentus usa um formato de *ticker* diferente da Yahoo Finance que possui um “.SA” ao final. Por exemplo o *ticker* da Vale (VALE3) é VALE3.SA na Yahoo.

Foi adicionada uma coluna ao dataframe para facilitar utilização dessa biblioteca.
Cada usuário deverá adequar o filtro ao seus interesses e propósitos.

```
acoes_filtradas.loc[:,tickerYfinance]=acoes_filtradas.index+'.SA'
```

5 - Eliminando Ações Sem Negócios

Para eliminar ações que eventualmente tenham sido deslistadas ou que não possuam negócios recentes, foi utilizada a função abaixo que usou os dados da Yahoo Finance para criar uma coluna com valor *booleano* para ações que tenham sido negociadas no último mês.

```
def obter_data_ultima_negociacao(ticker_symbol):
    try:
        ticker = yf.Ticker(ticker_symbol)
        historico = ticker.history(period="1mo")  # Ajuste o período, se necessário

        # Verificar se o histórico não está vazio
        if not historico.empty:
            ultima_data = historico.index[-1]  # Última data no índice do DataFrame
            return True  # Retorna a data (sem o horário)
        else:
            return False
    except Exception as e:
        print(f"Erro ao obter histórico para {ticker_symbol}: {e}")
        return False

acoes_filtradas['recente']=acoes_filtradas[tickerYfinance].apply(obter_data_ultima_negociacao)
acoes_filtradas=acoes_filtradas[acoes_filtradas['recente']==True]
```

A função foi aplicada e os valores filtrados para ‘verdadeiro’, ou seja, com negócios nos últimos 30 dias.

6 - Adequações adicionais

Para facilitar a futura visualização dos dados foi adicionada ao *dataframe* a informação do nome das empresas a que se refere a ação:

```
def obter_nome_empresa(ticker):
    try:
        info = yf.Ticker(ticker).info
        return info.get("longName", "Nome não disponível")
    except Exception as e:
        return "Erro na consulta"

acoes_filtradas['Empresa'] = acoes_filtradas[tickerYfinance].apply(obter_nome_empresa)
```

Adicionou-se após o setor de atuação da empresa:

```
def get_sector(ticker):  #função para obter o setor de ação constante na Yahoo Finance
    try:
        return (yf.Ticker(ticker).info['sector'])
    except Exception as e:
        return None
acoes_filtradas.loc[:,'Setor']=acoes_filtradas[tickerYfinance].apply(get_sector)
```

Para futura utilização nos gráficos foi incluída a informação da variação da ação em 52 semanas:

```
def get_52WeekChange(ticker):
    try:
        return (yf.Ticker(ticker).info['52WeekChange'])
    except Exception as e:
        return None

acoes_filtradas.loc[:,'variacao52Semanas']=acoes_filtradas[tickerYfinance].apply(get_52WeekChange)
```

7 – Salvando os resultados

Uma vez realizadas as filtragens e adição de informações é importante salvar o *dataframe* para um arquivo, pois isso facilitará a geração dos gráficos em ter que repetir todas as etapas.

Além disso, a Yahoo Finance pode limitar as consultar se forem feitas muitas requisições:

```
acoes_filtradas.to_csv('acoes_filtradas_backup.csv')
```

Neste projeto foi utilizado o Google Colab, assim após salvar os resultado para o arquivo em csv, faça download para evitar perde-lo.

Após o download salve o arquivo em um local adequado para utilização futura.
No caso deste projeto, foi utilizado o Google Drive com compartilhamento público.

Dessa forma, pode-se utilizar, por exemplo o *Streamlit Community Cloud* para rodar o aplicativo em qualquer navegador, sem limitações de acesso a um arquivo local.

8 – Ajustando o link do arquivo

Caso a sua opção seja salvar a planilha no Google Drive, ao copiar o link você deve fazer uma alteração para permitir que ela lida pelo script:

```
https://drive.google.com/file/d/1SJFgW5PbFbxwGgx-GG6dBq2-pS58IoSH/view?usp=drive_link
```

A parte em vermelho deve ser apagada e `/file/d` deve ser substituída por `uc?id=`.

O resulta ficaria dessa forma:

```
https://drive.google.com/uc?id=1SJFgW5PbFbxwGgx-GG6dBq2-pS58IoSH
```

9 - Preparando o app Streamlit

Nessa etapa final, vamos criar o app do Streamlit para gerar os gráficos desejados.

Apesar de ser apresentado em partes segmentadas, o script deverá ser executado de uma só vez.

Caso utilize Google Colab ou Jupyter Notebook, o conteúdo do script deverá estar numa única célula.

Parte1: criação do arquivo app.py (aplicativo Streamlit e inicialização das bibliotecas.

```
with open("app.py", "w") as f:
    f.write("""
import streamlit as st
import plotly.express as px
import pandas as pd
import numpy as np
import yfinance as yf
from datetime import datetime, timedelta
```

9.1 – Acessando a planilha de dados

Parte 2: Agora, o arquivo com os dados das empresas que foi filtrado e manipulado anteriormente será carregado.
As informações em percentual (Dividend Yield e Variação de 52 semanas, foram ajustadas para duas casas decimais.

```
acoes_filtradasgraf = pd.read_csv('https://drive.google.com/uc?id=1SJFgW5PbFbxwGgx-GG6dBq2-pS58IoSH')
acoes_filtradasgraf['dy'] = (acoes_filtradasgraf['dy'] * 100).round(2)
acoes_filtradasgraf['variacao52Semanas'] = acoes_filtradasgraf['variacao52Semanas'].fillna(0)
acoes_filtradasgraf['variacao52Semanas'] = (acoes_filtradasgraf['variacao52Semanas'] * 100).round(2)
```

Parte 3: Aqui é criada a função que obtém a maior cotação da ação no ano e calcula a diferença percentual em relação à cotação de hoje.

```
def calcular_distancia_topo(ticker):
    try:
        ativo = yf.Ticker(ticker)
        preco_topo = ativo.fast_info.year_high
        preco_atual = ativo.fast_info.last_price
        return round(((preco_atual - preco_topo) / preco_atual) * 100, 2)
    except Exception as e:
        st.warning(f"Erro ao calcular distância para o topo de {ticker}: {e}")
        return None
acoes_filtradasgraf['distanciaTopo'] = acoes_filtradasgraf['tickeryf'].apply(calcular_distancia_topo)
```

9.2 – Dados históricos e médias

Parte 4: Nessa etapa foi criada a função que obtém os dados históricos para o gráfico de linhas. Aqui também é calculada a média móvel de 20 e 50 dias.

```python
def obter_dados_historicos(ticker):
    try:
        data_fim = datetime.today()
        data_inicio = data_fim - timedelta(days=365)
        dados = yf.download(ticker, start=data_inicio.strftime("%Y-%m-%d"), end=data_fim.strftime("%Y-%m-%d"))
        dados['Fechamento'] = round(dados['Close'], 2)
        dados['Média 20 dias'] = round(dados['Fechamento'].rolling(window=20).mean(), 2)
        dados['Média 50 dias'] = round(dados['Fechamento'].rolling(window=50).mean(), 2)
        return dados
    except Exception as e:
        st.error(f"Erro ao obter dados para {ticker}: {e}")
        return None
```

Parte 5: Início da interface do Streamlit, definição do título do gráfico e o seletor para os setores das ações.

```python
def main():
    st.markdown(
        "<h2 style='text-align: center; font-size: 20px;'>Gráfico de bolhas)</h2>",
        unsafe_allow_html=True
    )

    setor_selecionado = st.selectbox(
        "Selecione o setor:",
        options=acoes_filtradasgraf['Setor'].unique(),
        index=0
    )
    df_filtrado = acoes_filtradasgraf[acoes_filtradasgraf['Setor'] == setor_selecionado]
```

9.4 – Geração do gráfico de bolhas

Parte 6: Criação do gráfico de bolhas, onde o eixo de X será o preço da ação sobre o valor patrimonial por ação, o eixo Y será a distância percentual da cotação de hoje para o preço máximo em um ano. O tamanho da bolha representa o Dividend Yield e a cor da bolha indica se ela está valorizada (azul), desvalorizada (vermelha), sendo a cor branca representando a estabilidade.

```
if not df_filtrado.empty:
        fig = px.scatter(
            df_filtrado,
            x='pvp',
            y='distanciaTopo',
            size='dy',
            color='variacao52Semanas',
            color_continuous_scale=px.colors.sequential.RdBu,
            color_continuous_midpoint=0.0,
            text='tickeryf',
            title=f"Ações do Setor: {setor_selecionado}",
            labels={
                'pvp': 'Preço sobre Valor Patrimonial',
                'distanciaTopo': 'Distância para o topo de 365 dias',
                'dy': 'Dividend Yield (%)',
                'variacao52Semanas': 'Variação das 52 semanas',
                'tickeryf':'Ticker'

            }
        )
```

9.5 – Atualizando o gráfico de bolhas

Parte 7: Atualização do ´gráfico com posicionamento do texto, adição dos rótulos dos eixos, escolha de um template para formatação e adição de uma linha tracejada para indicar ações que não tiveram variação (0,0%)

```
fig.update_traces(textposition='top center')

fig.update_layout(
    xaxis_title='Preço sobre Valor Patrimonial',
    yaxis_title='Distância para o topo de 365 dias',
    template='plotly_dark',
    shapes=[
        dict(type='line', x0=0, x1=1, y0=0, y1=0, xref='paper', yref='y', line=dict(color='red', dash='dash'))
    ]
)

st.plotly_chart(fig)
```

Uma breve explicação:

- Em X, ações que estiverem abaixo de 1, pode indicar subvalorização.
- Em Y, uma ação com valor inferior ao topo no último ano, poderia indicar oportunidade de valorização
- Bolhas maiores representam melhores pagamentos de dividendos.

9.6 – Criando o gráfico de linhas

Parte 8: Após identificar no gráfico de bolhas uma ação do seu interesse, o usuário, poderá selecionar uma ação para acompanhar o comportamento dela no último ano.

```
ticker_selecionado = st.selectbox(
            "Selecione uma ação para visualizar o desempenho:",
            options=df_filtrado['tickeryf'].unique()
        )

        if ticker_selecionado:
            dados_historicos = obter_dados_historicos(ticker_selecionado)
            if dados_historicos is not None:
                fig_variacao = px.line(
                    dados_historicos.reset_index(),
                    x='Date',
                    y=['Fechamento', 'Média 20 dias', 'Média 50 dias'],
                    title=f"Variação Cotação e Médias Móveis ({ticker_selecionado})",
                    labels={'value': 'Fechamento', 'Date': 'Data', 'variable': 'Indicadores'},

                    color_discrete_map={
                        'Fechamento': 'blue',
                        'Média 20 dias': 'green',
                        'Média 50 dias': 'red'
                    }
                )
```

Parte 9: Aqui são adicionadas as espessuras das linhas para dar mais destaque à cotação do ativo em análise:

```
fig_variacao.update_traces(selector=dict(name='Fechamento'), line=dict(width=1))  # Linha de variação mais espessa
fig_variacao.update_traces(selector=dict(name='Média 20 dias'), line=dict(width=0.5))  # Linha mais fina
fig_variacao.update_traces(selector=dict(name='Média 50 dias'), line=dict(width=0.8))  # Linha de espessura intermediária

st.plotly_chart(fig_variacao)
```

9.7 – Finalizando

Parte 10: Aqui será adicionado um **`else:`** que foi iniciado na em **`if not df_filtrado.empty`** e é finalizado o app do streamlit:

```
else:
        st.warning("Nenhuma ação encontrada para o setor selecionado.")
if __name__ == "__main__":
    main()
    """)
```

Após essa etapa haverá um arquivo nomeado como app.py no Google Colab. Você poderá baixa-lo e rodar na sua máquina a partir do prompt de comando e o aplicativo será aberto no seu navegador:

```
streamlit run app.py
```

10 – Acesso ao script completo

O projeto está finalizado e funcional.

Para acessar o script completo, para modificações e aperfeiçoamentos acesse o link abaixo:

https://colab.research.google.com/drive/1Asl-i_XpW5k21HEjZjIWJMMEJdoeGggE?usp=sharing

Agradecimentos

Este eBook apresentou um guia prático para a criação de um projeto interativo de análise de ações utilizando **Python** com as bibliotecas **Streamlit**, **Plotly** e **YFinance**. O projeto foi estruturado para permitir a construção de gráficos dinâmicos, como **gráficos de bolhas** e **gráficos de linhas**, facilitando a visualização de informações financeiras relevantes, como o preço sobre valor patrimonial (P/VP), distância do preço atual para o topo de 52 semanas e médias móveis de 20 e 50 dias.

Agora explore as possibilidade de personalize de acordo com os seus interesses.

Obrigado por consultar este e-book.