Comparador de ações

Olá! Temos ótimas notícias: seu chefe está muito feliz com o seu desempenho e você recebeu uma bela bonificação em dinheiro como recompensa!

Depois de pensar um pouco, você decidiu que não é hora de trocar de carro nem de renovar o guarda-roupa, e concluiu que é uma boa ideia aplicar esse dinheiro para o futuro.

Como a taxa SELIC está em um momento de baixa histórica, sua conclusão foi que vale mais à pena investir em uma empresa, comprando suas ações. Tenha em mente que o seu objetivo é de longo-prazo. Você não pretende praticar day trading, mas "esquecer" do dinheiro e resgatá-lo em sua aposentadoria.

Antes de tomar uma decisão tão importante, você decidiu fazer uma análise mais cuidadosa das suas empresas favoritas para ver qual ou quais oferecem mais segurança. E a ferramenta mais natural para isso são os seus conhecimentos adquiridos em seu curso de programação em Python!

Como enviar o programa para avaliação?

Na última aula liberada no Class (Funções com parâmetros variáveis e retornos múltiplos) há uma nova lista de exercícios. Essa lista contém um único exercício: este trabalho.

Copiar/colar um Jupyter Notebook seria muito trabalhoso. Ao invés disso, quando você tiver concluído completamente seu trabalho, aqui no Jupyter clique em FILE > DOWNLOAD AS > PYTHON (.py).

Abra o arquivo .py gerado no Bloco de Notas ou equivalente, copie todo seu conteúdo e cole como resposta do exercício no Class.

Fonte de dados: Alpha Vantage

Seu programa precisará de dados sobre as ações para ajudá-lo em sua tomada de decisão. Utilizaremos uma API para obter esses dados. Não se preocupe com detalhes sobre o que é uma API, estudaremos isso muito em breve. Por hora, vamos aceitar que é uma fonte de dados da internet.

A API escolhida para essa tarefa foi a Alpha Vantage. Ela é bastante completa, contendo mais de 20 anos de histórico sobre ações em bolsas ao redor do mundo. E o melhor: ela é *grátis*. Para utilizá-la, porém, é necessário fornecer seu endereço de e-mail e eles enviarão uma chave de acesso.

Faça seu registro em https://www.alphavantage.co/support/#api-key e atualize a variável "alphakey" na célula abaixo com a chave que você recebeu por e-mail. Lembre-se que essa chave é uma string. Não se esqueça de executar a célula após atualizar sua chave!

In [1]:
```
alphakey = 'XAYLROB7KRUTDSM2'
```

Vamos testar nossa chave? A célula abaixo irá mostrar algumas informações sobre o valor da PETR3.SA (Petrobrás ON) no dia 30/06/2021. Execute o código abaixo e veja se ele mostra corretamente os valores ou alguma mensagem de erro.

In [2]:
```
import requests
```

```
teste = requests.get('https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=PETR3.SA&apikey='+alphakey)
teste = teste.json()
print(teste['Time Series (Daily)']['2021-06-30'])
```

```
{'1. open': '29.6000', '2. high': '30.4900', '3. low': '29.4700', '4. close': '30.2900', '5. volume': '38464699'}
```

Preparando nossos dados

Você verá que mais para frente teremos alguns trechos de código pronto para auxiliá-lo. Esses trechos de código utilizam funções, mas aqui entra a "pegadinha": várias dessas funções estão em branco e você deverá elaborar sua lógica!

Seu primeiro trabalho será ajudar a organizar os dados. Nossos trechos de código pronto fornecerão para você um dicionário contendo as informações por data. Cada data é uma chave, e o seu valor será outro dicionário contendo as diferentes informações sobre cada dia (valor de abertura, fechamento, máximo, mínimo e volume de negociações). Veja um exemplo desse dicionário abaixo.

```
{'2021-06-30':
    {'1. open': '29.6000',
     '2. high': '30.4900',
     '3. low': '29.4700',
     '4. close': '30.2900',
     '5. volume': '38464700'},
 '2021-06-29':
    {'1. open': '29.3200',
     '2. high': '29.7600',
     '3. low': '29.0000',
     '4. close': '29.6800',
     '5. volume': '18320100'},
 '2021-06-28':
    {'1. open': '29.4900',
     '2. high': '29.5600',
     '3. low': '29.0200',
     '4. close': '29.2800',
     '5. volume': '11133400'},
  ...
}
```

A função formataInfos esboçada na célula abaixo irá receber um dicionário como esse exibido acima e deverá retornar duas listas:

- A primeira deve conter todas as datas, em ordem crescente. As datas devem ser mantidas no formato que estão no dicionário: uma string no formato 'aaaa-mm-dd'.
- A segunda deve conter todos os valores de fechamento. Cada valor deverá estar no índice correspondente ao de sua data. Os dados devem ser convertidos para float. Para o exemplo acima, suas listas ficariam:
  `datas = ['2021-06-28', '2021-06-29', '2021-06-30']`
  `valores = [29.28, 29.68, 30.29]`

In [3]:
```
def formataInfos(dicionario):
```

```
    # insira sua lógica aqui
    datas = []
    valores = []

    # Retorna todas as datas de fechamento na lista 'datas'
    # e os respectivos valores de fechamento na lista 'valores'.
    # Estudando vários dicionários recebidos verificamos que sempre estão em ordem decrescente, ou seja, as datas mais
    # recentes no início. Assim, como queremos que as datas mais recentes estejam mais à direita no eixo X do gráficos
    # que serão apresentados, invertemos as listas ao sempre inserir dados na posição 0, "empurrando" os dados anteriormente
    # alimentados para "direita"
    #
    # Poderia utilizar um sort na lista "datas", para então buscar o valor de cada chave num segundo passo.
    # Aqui, sem tratar o formato de data, estamos supondo que sejam sempre yyyy-mm-dd, ou seja, para uma ordenação segura
    # seria necessário também converter o valor string em data e reformatá-lo para depois ordenar. Logo, optei por manter a
    # solução simples, para os propósitos do exercício.
    for info in dicionario:
        datas.insert(0,info)
        valores.insert(0,float(dicionario[info]['4. close']))

    return datas, valores
```

Se estivermos comparando ações de uma empresa mais recente com uma empresa mais antiga, teremos um problema: pode acontecer de termos mais datas para uma das empresas do que para a outra. Isso irá comprometer o funcionamento do nosso programa - por exemplo, o gráfico da mais nova pode ficar deslocado.

A função preencheZeros esboçada abaixo recebe 4 listas: datas de uma empresa, valores dessa mesma empresa, datas da outra empresa, valores dessa outra empresa. Ela deve testar se as empresas possuem a mesma quantidade de dados. Caso alguma delas possua MENOS, sua lista de datas deve ser "completada" com as datas que a empresa mais antiga possui e ela não. Os valores correspondentes na lista de valores serão preenchidos com 0.

Exemplo de entrada:

```
datas1 = ['2021-06-28', '2021-06-29', '2021-06-30']
valores1 = [29.28, 29.68, 30.29]
datas2 =  ['2021-06-29', '2021-06-30']
valores2 = [31.43, 30.98]
```

Saída correspondente:

```
datas1 = ['2021-06-28', '2021-06-29', '2021-06-30']
valores1 = [29.28, 29.68, 30.29]
datas2 =  ['2021-06-28', '2021-06-29', '2021-06-30']
valores2 = [0.0, 31.43, 30.98]
```

Note que você modificará diretamente as listas recebidas, não sendo necessário retornar nada.

In [4]:
```
def preencheZeros(datas1, valores1, datas2, valores2):
```

```
# Utilizando duas listas de listas e for para processar os dois pares recebidos (datas x valores)...
listasDatas   = [datas1, datas2]
listasValores = [valores1, valores2]

# ...vamos preencher duas listas de tuplas, que darão suporte para a geração dos resultados finais.
tuplas = [[],[]]

# A opção por tuplas minimiza riscos de tratar datas e valores separadamente, afinal, um descasamento do par data x valor
# é potencialmente desastroso e difícil de detectar.

# Processa o par de listas 0 na primeira iteração e o par de listas 1 na segunda.
for lista in range(2):

    # Aponta para o outro par de listas (0 aponta para o par 1 e 1 aponta para o par 0)
    outraLista = 1 - lista

    # Percorre o par de listas data x valor (utiliza range porque é necessário manipular por índice)
    for i in range(len(listasDatas[lista])):
        # Data e valor na próxima posição da lista
        data  = listasDatas[lista][i]
        valor = float(listasValores[lista][i])
        # Cria tupla e armazena na lista em processamento (0 ou 1)
        tupla = data, valor
        tuplas[lista].append(tupla)

        # Verifica se a data existe na outra lista de datas.
        # Uma exceção ValueError ocorre quando a data não existe...
        try:
            indice = listasDatas[outraLista].index(data)
        except ValueError:
            # ... e assim a cadastramos com valor 0.0 para, ao final, termos ambas listas com mesmo tamanho
            tupla = data, 0.0
            tuplas[outraLista].append(tupla)

# Nesse ponto as duas listas de tuplas estão preenchidas
# Cada lista de tuplas é processada para garantir a ordem crescente de datas e gerar as listas separadas esperadas
# como retorno
for lista in range(2):

    tuplas[lista] = sorted(tuplas[lista], key=lambda item: item[0])

    listasDatas[lista].clear()
    listasValores[lista].clear()

    for item in tuplas[lista]:
        listasDatas[lista].append(item[0])
        listasValores[lista].append(item[1])
```

Se você já viu gráficos mostrando o valor de ações ao longo do tempo, certamente já percebeu que eles são extremamente ruidosos. Isso é causado por pequenas oscilações de um dia pro outro, de uma hora para outra e até mesmo de um minuto para o outro. Essas oscilações ocorrem por movimentos naturais de compra e venda.

Como essas oscilações são mais ou menos aleatórias e ocorrem com a mesma frequência para cima e para baixo, seu efeito para análises de longo prazo pode ser desprezado. Estamos mais interessados na tendência geral de crescimento ou queda ao longo de muitos meses ou anos, e não em pequenas variações de humor do mercado.

Podemos suavizar nossa curva utilizando um filtro de média móvel. Esse filtro define o valor de um ponto na curva como sendo a média dos "n" pontos anteriores. O número "n" de pontos utilizado para a média está relacionado ao grau de atenuação do ruído. Quanto mais alto esse valor, maior a atenuação - mas se for alto demais, pode comprometer o formato geral da curva.

O gráfico abaixo ilustra a média móvel em ação. Note que em azul escuro temos uma curva bastante "rabiscada", e em azul mais claro há uma curva mais suave, mostrando o formato da curva original sem as oscilações aleatórias.

Vamos suavizar as curvas de nossa base de dados?

A função mediaMovel abaixo recebe uma lista de valores e um fator "n". Ela deve *retornar* uma lista contendo o sinal filtrado.

Atenção: se você tem n=10 e está calculando a amostra no índice 5, você NÃO deve pegar os índices negativos para completar 10 amostras. Pegue apenas os índices de 0 até 4. O comecinho da curva irá se distorcer um pouco, mas ela rapidamente converge para a forma da curva original.

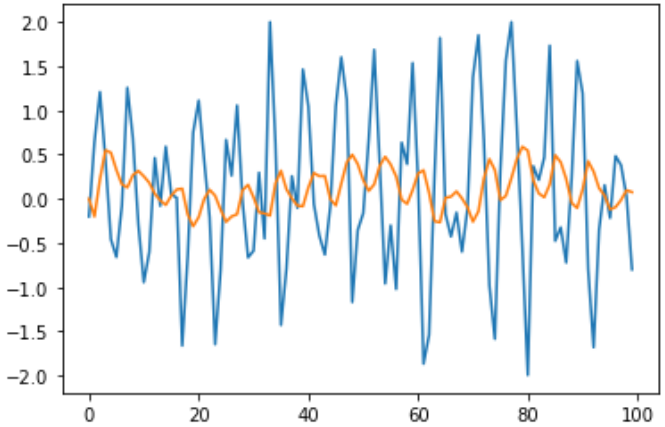
In [5]:
```
def mediaMovel(valores, n):
```

```
    filtrado = []

    # insira sua lógica aqui

    # Devemos gerar elementos a cada N itens da lista de valores, desconsiderando índices negativos
    # Cada elemento representa a média dos N itens coletados, sempre anteriores ao ponto atual (indice na lista)
    for i in range(len(valores)):
        soma = 0.0
        qtd = 0
        # Percorre os N elementos anteriores ao índice i atual
        for j in range(i-n, i):
            # Descarta os índices negativos
            if (j >= 0):
                soma += valores[j]
                qtd += 1
        # Trata pontos com quantidade 0 e 1 de ítens, onde a divisão não deve ocorrer
        if (qtd > 1):
            soma = soma / qtd
        # Adiciona o novo ponto calculado na lista a retornar
        filtrado.append(soma)

    return filtrado
```

Vamos testar nossa média móvel? Execute o código abaixo. Se sua média móvel estiver correta, você deve ver um gráfico com comportamento semelhante ao gráfico acima: uma curva "rabiscada" e outra suave com o mesmo formato aproximado.

In [6]:
```
from matplotlib import pyplot as plt
import random
import math

sinal = [math.sin(x) + (random.randint(-10,10)/10) for x in range(100)]
suave = mediaMovel(sinal, 10)

plt.plot(sinal)
plt.plot(suave)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x18e0f515a60>]
```



Extraindo informações

Estamos indo bem! Até o momento você já se registrou para utilizar uma API, separou os dados que utilizaremos, preencheu lacunas nos dados e filtrou o ruído da informação. Hora de começarmos a levantar informações que possam ser úteis para a tomada de decisões.

Vamos começar pela mais fácil: máximos e mínimos.

A função maiorData abaixo recebe uma lista de datas e uma lista com os respectivos valores e retorna o maior de todos os valores e a respectiva data, no formato (data, valor).

A função menorData é semelhante, mas para o menor valor.

In [7]:
```python
def maiorData(datas, valores):
    data = datas[0]
    valor = valores[0]

    # insira sua lógica aqui

    # Retorna a data em que ocorreu o maior valor da série
    for indice in range(1,len(datas)):
        if valores[indice] > valor:
            data  = datas[indice]
            valor = valores[indice]

    return data, valor

def menorData(datas, valores):
    data = datas[0]
    valor = valores[0]

    # insira sua lógica aqui

    # Retorna a data em que ocorreu o menor valor da série
    for indice in range(1,len(datas)):
```

```
        if valores[indice] < valor:
            data  = datas[indice]
            valor = valores[indice]

    return data, valor
```

Outra informação que pode ser útil é a amplitude. Ela pode mostrar o tamanho das surpresas (positivas ou negativas) que outros investidores naquele ativo já tiveram. A amplitude será a diferença entre o maior e o menor valor. Mas como saber se essa oscilação foi boa ou ruim?

- se a data do maior valor foi superior à do menor valor, a amplitude será positiva (max - min)
- se a data do maior valor foi inferior à do menor, a amplitude será negativa (min - max)

A função amplitude irá receber, respectivamente, a data do maior valor, o maior valor, a data do menor valor e o menor valor, e deverá retornar o valor correto da amplitude (atente-se à ordem para fazer a subtração corretamente conforme a regrinha acima).

In [8]:
```
def amplitude(maiordata, maiorvalor, menordata, menorvalor):

    # insira sua lógica aqui

    # Retorna a amplitude entre o maior e o menor valor
    if maiordata >= menordata:
        amp = maiorvalor - menorvalor
    else:
        amp = menorvalor - maiorvalor

    return amp
```

Uma ferramenta estatística que pode nos auxiliar a entender a tendência de longo prazo (e desprezando muitos detalhes...) é a regressão linear.

Na regressão linear nós partimos de um conjunto de pontos de dados e tentamos gerar a reta que melhor se aproxime desses pontos.

Naturalmente, não podemos levar a regressão linear totalmente ao pé da letra para curvas complexas, como a oscilação dos valores de ações. Mas ela pode nos dar uma indicação se o comportamento geral da curva foi de subida ou de queda.

Podemos escrever uma reta na forma:

$$y = ax + b$$

Onde "a" é o coeficiente angular da reta (relacionado ao grau de inclinação da reta), e b indica o valor de y quando x = 0.

Quanto maior o valor de "a", mais rápido a reta cresce - bastante desejável em nosso caso! Valores negativos de "a" indicam reta decrescente.

A função abaixo utiliza uma função pronta para calcular os termos "a" e "b" da reta, além de gerar uma lista de valores de x. Seu trabalho será simples: gere uma lista contendo o valor de "y" para cada valor de "x". Esses são os valores da reta obtida e serão usados para traçar o gráfico.

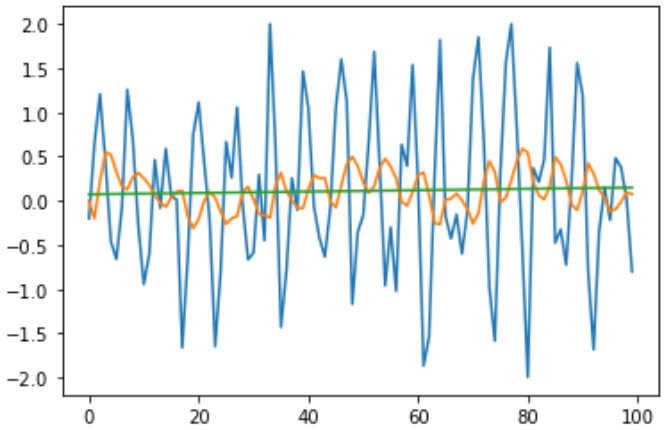
A função retornará, na ordem, a, b, x e y.

In [9]:
```
from scipy.stats import linregress

def regressaoLinear(valores):
    x = list(range(len(valores)))
    y = []

    stats = linregress(x, valores) # calculamos a regressão linear
    a = stats.slope # coeficiente angular
    b = stats.intercept # onde intercepta o eixo y

    # agora é com você: calcule os valores de y para cada x da reta!
    # insira sua lógica aqui

    # Calcula cada valor de Y e preenche a lista a ser retornada
    for valorX in x:
        y.append(a * valorX + b)

    return a, b, x, y
```

Vamos testar nossa regressaoLinear? Lembra dos dados que testamos nossa média móvel lá em cima? Rode o trecho abaixo e veja a regressão linear daqueles dados.

In [10]:
```
a, b, x, y = regressaoLinear(suave)

plt.plot(sinal)
plt.plot(suave)
plt.plot(y)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x18e1077fd60>]
```



Resultados

Agora que você já suou a camisa implementando diversas funções, chegou a hora de ver o resultado.

Você está recebendo de presente, 100% pronta, uma função que recebe uma lista de símbolos de ações (como 'PETR3.SA', 'MGLU3.SA' etc), mostra os valores de máximo e mínimo de cada uma, a maior amplitude detectada, informa se a oscilação foi positiva ou negativa e no final plota os gráficos comparando seu desempenho. Não modifique essa função!

In [11]:
```
def resultados(acoes):
    dicionario = {}
    plt.rcParams['figure.figsize'] = [12, 8]
    plt.rcParams['figure.dpi'] = 100
    for a in acoes:
        try:
            print('Baixando e processando os dados de', a+'...')
            resposta = requests.get('https://www.alphavantage.co/query?function=TIME_SERIES_MONTHLY&symbol='+a+'&apikey='+alphakey)
            resposta = resposta.json()

            # separar datas e valores
            print('Formatando...')
            datas, valores = formataInfos(resposta['Monthly Time Series'])

            # maior, menor e amplitude:
            maiordata, maiorvalor = maiorData(datas, valores)
            menordata, menorvalor = menorData(datas, valores)
            amp = amplitude(maiordata, maiorvalor, menordata, menorvalor)

            # preenche com zeros se estiver faltando algo
            # (original)
            #print('Preenchendo lacunas...')
            #for chave in dicionario:
            #    preencheZeros(datas, valores, dicionario[chave]['datas'], dicionario[chave]['valores'])
            #    preencheZeros(datas, valores, dicionario[chave]['datas'], dicionario[chave]['movel'])

            # preenche com zeros se estiver faltando algo
            # Ajustado
            print('Preenchendo lacunas...')
            for chave in dicionario:
                datasOriginal = dicionario[chave]['datas'].copy()
                preencheZeros(datas, valores, dicionario[chave]['datas'], dicionario[chave]['valores'])
                preencheZeros(datas, valores, datasOriginal, dicionario[chave]['movel'])

            # calcula a média móvel
            print('Calculando média móvel...')
            movel = mediaMovel(valores, 10)

            # calcula a regressão linear da média móvel
            m, b, x, y = regressaoLinear(movel)
            plt.plot(datas, valores)
            plt.plot(datas, movel)
            plt.plot(datas, y)
            plt.title(a)
            plt.xticks(rotation=45, fontsize='xx-small')
            plt.xlabel('Data')
            plt.ylabel('Valor')
            plt.legend(['Valor bruto', 'Média móvel', 'Regressão Linear'])
```

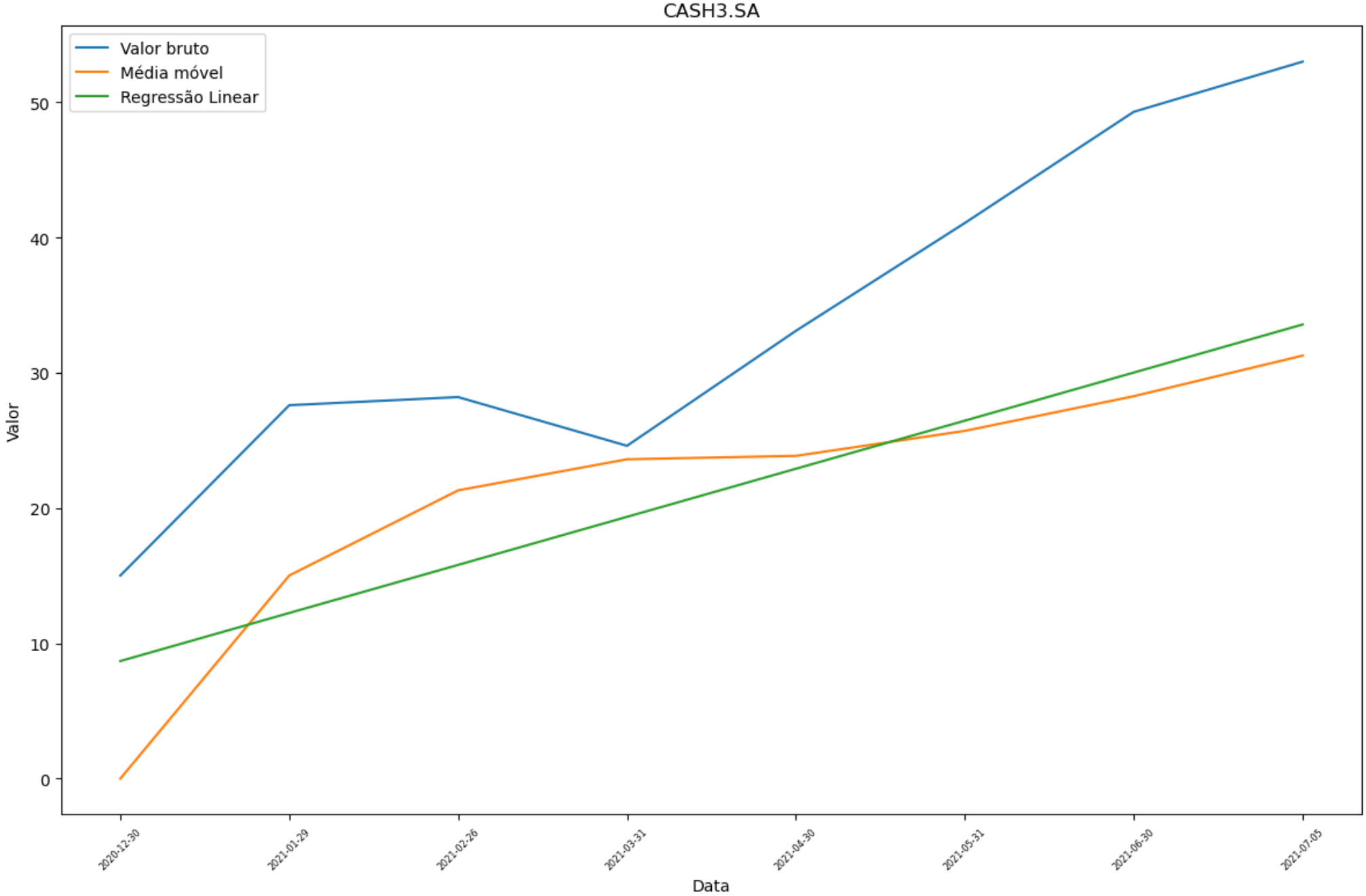
```
            plt.tight_layout()
            plt.show()

            # adiciona dados ao dicionario:
            dicionario[a] = {'datas':datas, 'valores':valores, 'movel':movel, 'm':m,
                             'maiordata':maiordata, 'maiorvalor':maiorvalor, 'menordata':menordata, 'menorvalor':menorvalor,
                            'amp':amp}

        except:
            print('Houve um problema ao trabalhar com a ação', a)

    # gerando informações comparativas
    for chave in dicionario:
        print(chave)
        print('Maior valor foi {}, atingido em {}.'.format(dicionario[chave]['maiorvalor'], dicionario[chave]['maiordata']))
        print('Menor valor foi {}, atingido em {}.'.format(dicionario[chave]['menorvalor'], dicionario[chave]['menordata'],))
        print('A maior amplitude foi',dicionario[chave]['amp'])
        plt.plot(dicionario[chave]['datas'], dicionario[chave]['movel'])

    plt.xlabel('Data')
    plt.ylabel('Valor')
    plt.xticks(rotation=45, fontsize='xx-small')
    plt.title('Comparativo entre ativos')
    plt.legend(acoes)
    plt.tight_layout()
    plt.show()
```

Utilize a célula abaixo agora para perguntar quantas ações o usuário gostaria de comparar. Leia os nomes das ações pelo teclado, monte uma lista e passe para a função resultado.

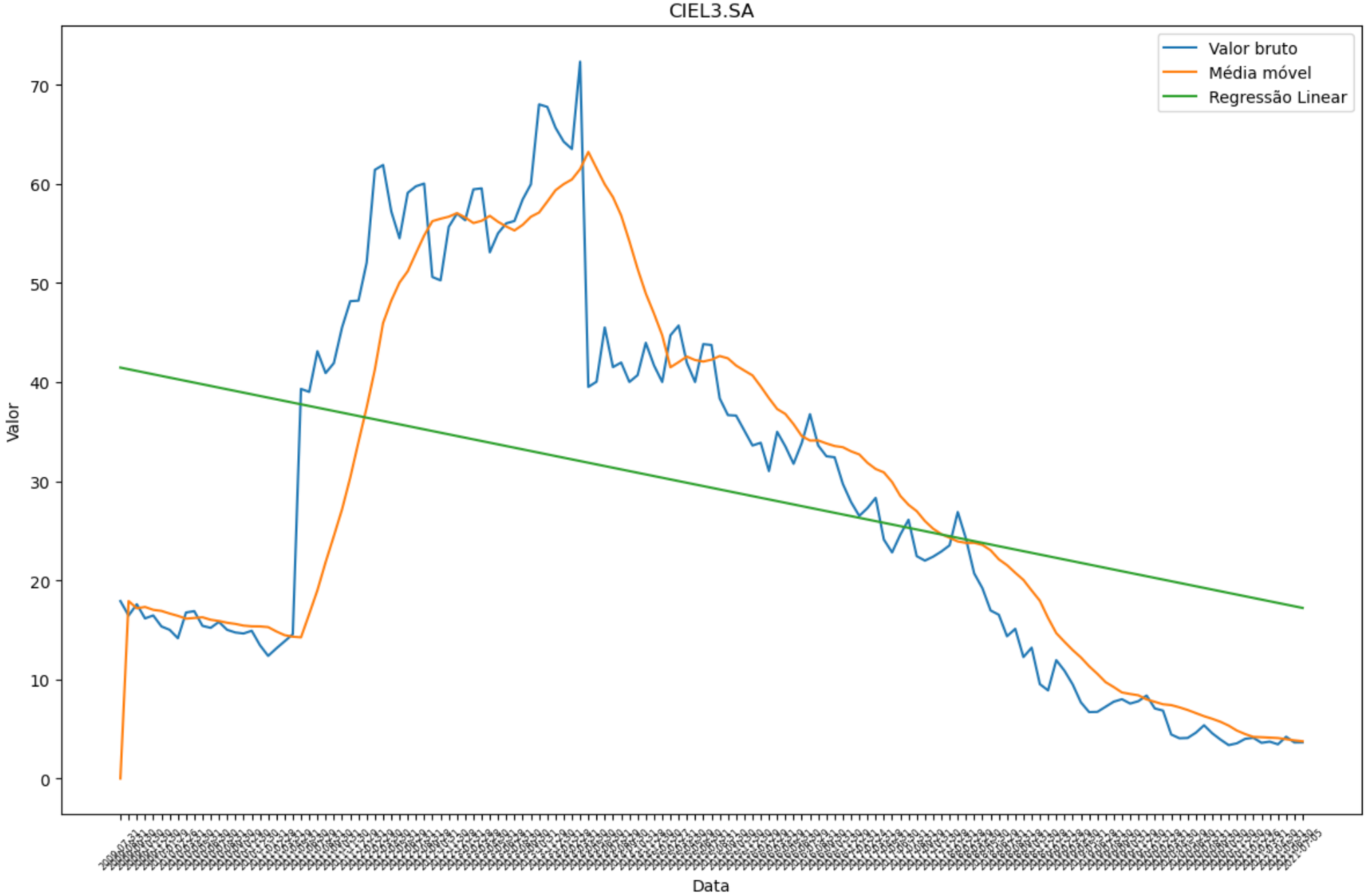
In [13]:
```
# insira sua lógica aqui!

# Solicita quantas ações o usuário deseja consultar e comparar
qtd = 0
while True:
    try:
        # Um valor numérico superior a 0 para continuar ou 0 para encerrar
        qtd = int(input("Quantas ações quer comparar (0 para nenhuma)? "))
        if qtd < 0:
            # Valores negativos não devem ser aceitos
            raise
        break
    except:
        # Ops! Usuário tem que corrigir o valor informado
        print("Erro: Por favor informe um número positivo!")

# Usuário selecionou uma quantidade de ações
if qtd > 0:
    # Solicita o código de cada ação desejada e cria uma lista com esses valores
    listaAcoes = []
    for n in range(qtd):
        listaAcoes.append(input("Informe código da ação "+str(n+1)+": "))
```

```
    # Executa o processamento sobre a lista de ações informada
    resultados(listaAcoes)

print("Fim de execução.")
```

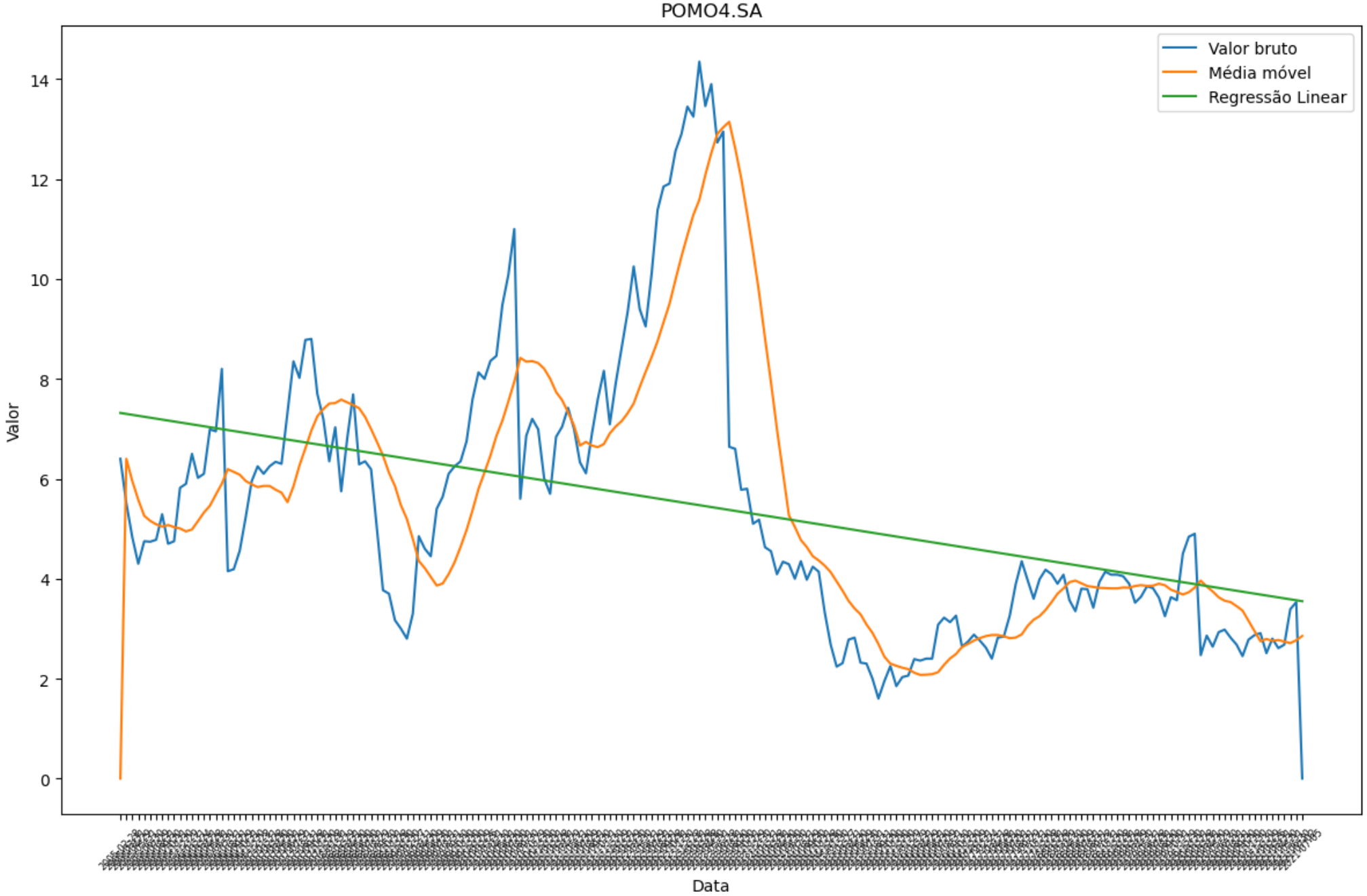
```
Quantas ações quer comparar (0 para nenhuma)? 3
Informe código da ação 1: CASH3.SA
Informe código da ação 2: CIEL3.SA
Informe código da ação 3: POMO4.SA
Baixando e processando os dados de CASH3.SA...
Formatando...
Preenchendo lacunas...
Calculando média móvel...
```



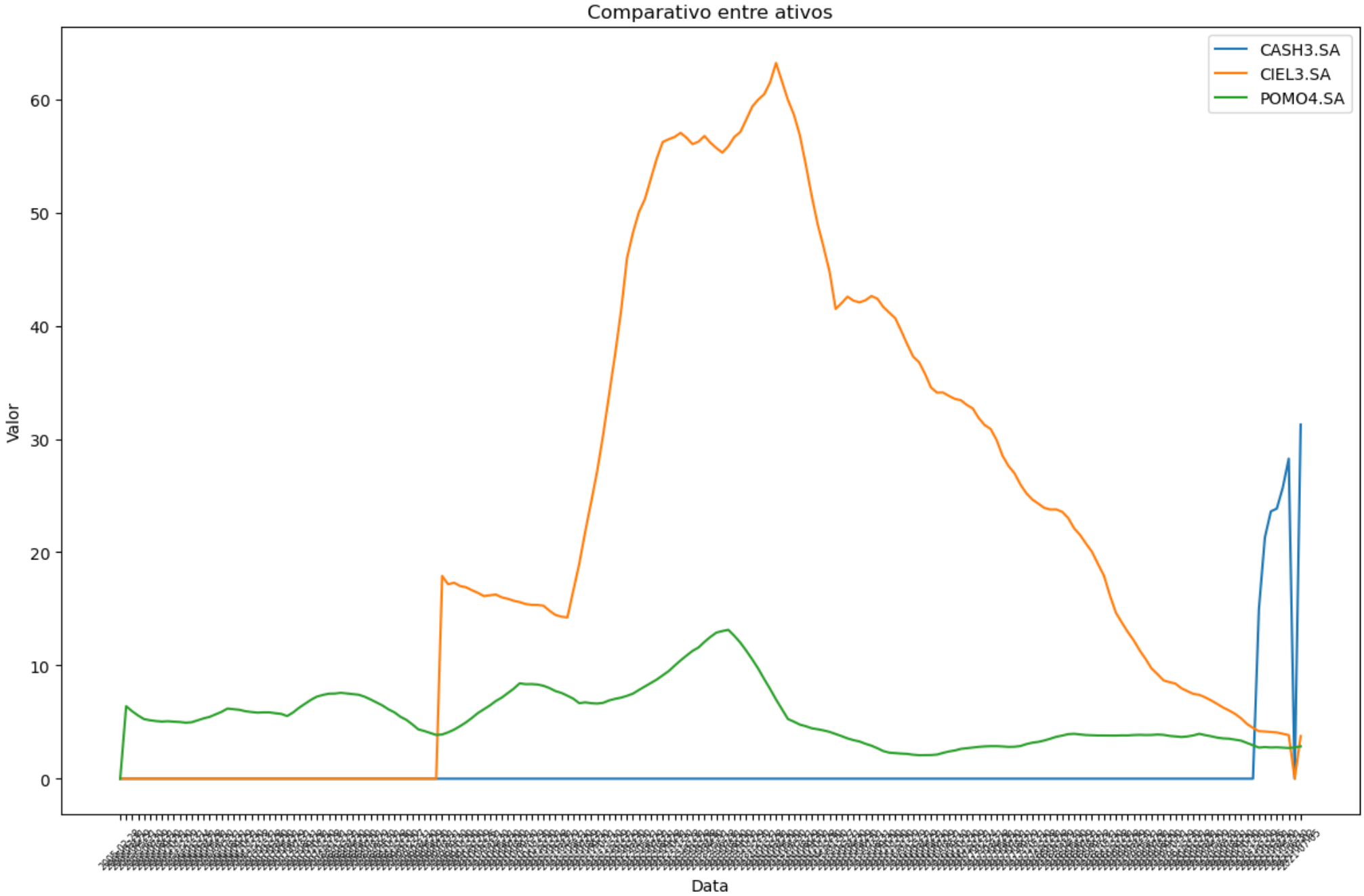

```
Baixando e processando os dados de CIEL3.SA...
Formatando...
Preenchendo lacunas...
Calculando média móvel...
```




```
Baixando e processando os dados de POMO4.SA...
Formatando...
Preenchendo lacunas...
Calculando média móvel...
```


POMO4.SA
Valor bruto
Média móvel
Regressão Linear
Valor
Data
0
2
4
6
8
10
12
14

CASH3.SA
Maior valor foi 53.0, atingido em 2021-07-05.
Menor valor foi 15.01, atingido em 2020-12-30.
A maior amplitude foi 37.99
CIEL3.SA
Maior valor foi 72.31, atingido em 2014-03-31.
Menor valor foi 3.37, atingido em 2020-10-30.
A maior amplitude foi -68.94
POMO4.SA
Maior valor foi 14.35, atingido em 2013-03-28.
Menor valor foi 1.6, atingido em 2015-09-30.
A maior amplitude foi -12.75




```
Fim de execução.
```

A célula abaixo é uma célula markdown. Fique à vontade para escrever comentários e observações - por exemplo, analisando os gráficos que você gerou em seus testes, você concorda ou discorda com a análise proposta? Por que?

Isso não vale nota, não tenha medo de fazer uma análise "certa" ou "errada"!

AMBP3 - Ambipar

A ação MicroCap de 06/2021.

Ambipar é uma ação bastante nova (08/2020), como podemos verificar no gráfico da ação, e pela extensa linha azul no valor 0 do eixo X do gráfico de comparação. A escolhi por ter estudado seu histórico recentemente. Como toda nova ação, ela inicia com um ticket médio razoável, que atrai os investidores iniciais. Nos seis meses iniciais praticamente não houve valorização expressiva dessa ação, porém um renomado analista vinha avisando sobre o alto potencial de valorização. Em abril/21 a ação foi reconhecida pelo mercado, e no periodo compreendido entre maio/21 e junho/21 houve uma expressiva valorização, como expresso em seu gráfico. É uma ação em tendência de alta, como aponta a linha de regressão apresentada.

PETR3 - Petrobrás

Uma das referências da Bolsa.

Podemos constatar que altas e baixas expressivas ocorreram nos anos iniciais, certamente proporcionando altos ganhos e também expressivas perdas aos acionistas. A Petrobrás, em particular, sofre pelas intervenções governamentais usuais. Houve um período longo de queda progressiva, pelo barateamento dessa commodity no mercado mundial, mas o aumento das exportações e procura por proteção decorrentes da pandemia têm inclufienciado positivamente os valores desses ativos. A linha de tendência do gráfico descreve o longo período de desvalorização, e deve alterar para uma tendência de crescimento, sobretudo se os temores envolvendo a pandemia persistirem.

Gráfico de Comparação

O gráfico apresenta perfeitamente a natureza de novidade da ação da Ambipar, e os movimentos da ação de petróleo, bem mais antiga. As cotações mês-a-mês são coerentes que a realidade, comprovando que a lógica de plotagem está correta. O gráfico comprova o preenchimento de períodos sem valores para ações com menor tempo de vida.

In [ ]: