Finanças 3: APS 2

Antonio Mendes Ehlers e Bruno Weber Maurer

**Resumo** — Esse trabalho visa a estruturação de um código em Python que permita calcular a probabilidade de Default de uma empresa através da fórmula de Merton, utilizando a volatilidade implícita das opções de call da empresa, e informações gerais (extraídas via Webscraping), tal como o total de ativos, passivos e o prazo médio da dívida da empresa.

—————————— ◆ ——————————

# 1 Parte 1 - Teórica

# 

# 2 Parte 2 - Prática

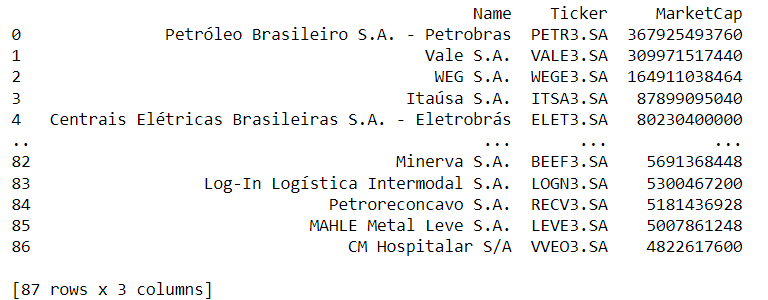
Para realizar a tarefa acima, é necessário que, primeiro, importemos as bibliotecas necessárias para a tarefa, visto que nosso grupo decidiu realizar a extração dos dados via webscrapping não foi utilizado nenhuma planilha no Excel para essa primeira parte do trabalho. Extraímos os tickers das principais empresas “large cap” do Brasil, através de um webscrap no site da TradingView, utilizando o BeautifulSoup.

Assim, extraímos os dados de Market Capitalization para todas as empresas e, através dos tickers que conseguimos anteriormente, mesclamos a tabela aos nomes completos das empresas.



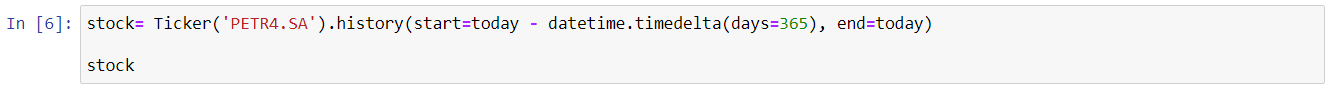


Em que conseguimos o seguinte resultado parcial:



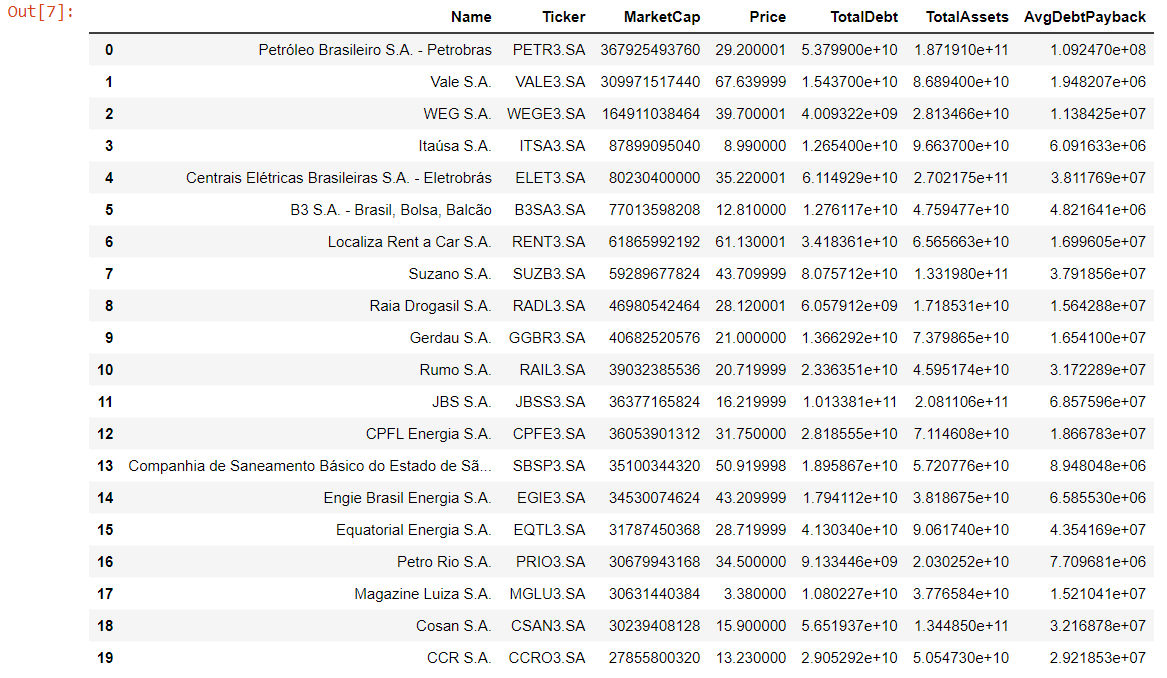
Em seguida, extraímos os dados de preço para cada Ticker, assim, adicionando tudo à tabela acima, em conjunto com os dados de total de ativos, passivos e o prazo médio da dívida:

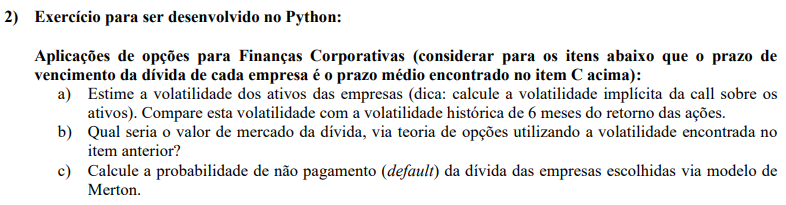






Assim, conseguimos o resultado completo da primeira parte prática da atividade:



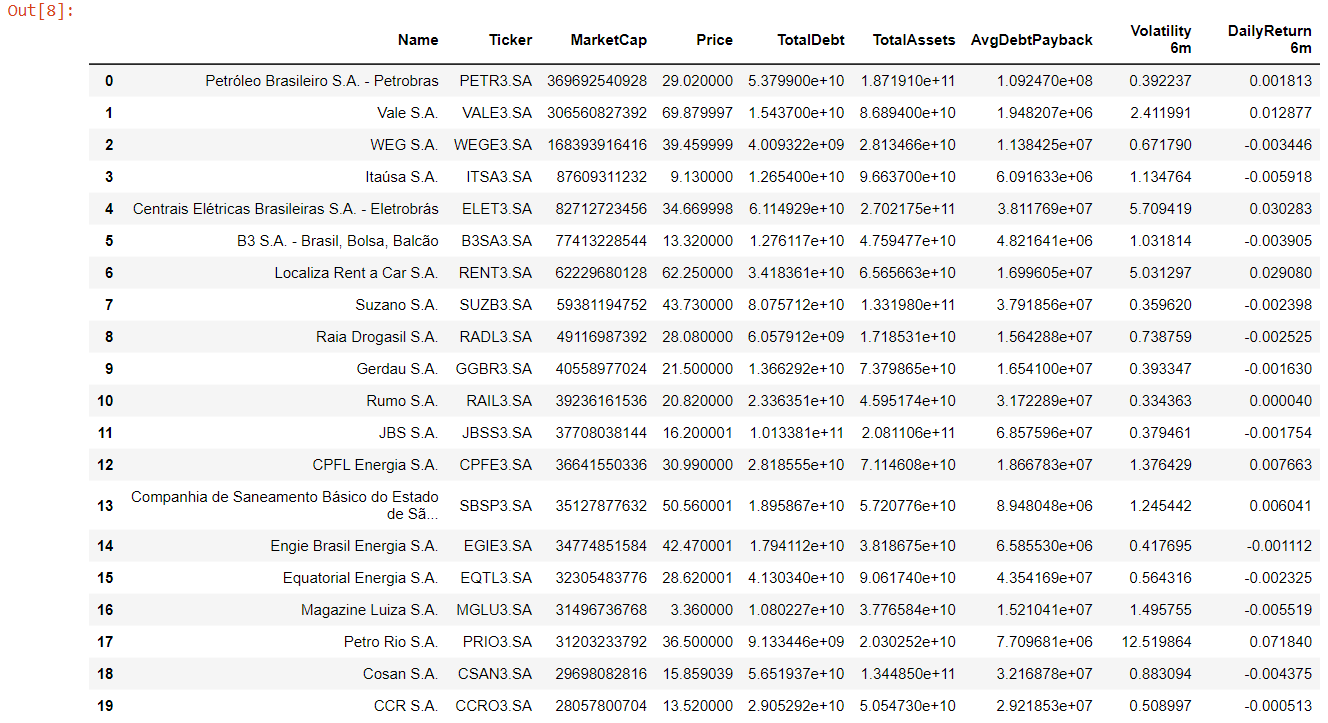


Calculamos a volatilidade histórica do ativo (de 6 meses) bem como os retornos médios diários através do código abaixo:





Resultando na tabela abaixo:



O código abaixo realiza a raspagem (scraping) de opções financeiras de um site chamado "Status Invest". Ele coleta dados de opções de diferentes ativos com base em uma lista de tickers fornecida.

1. É importada a biblioteca cloudscraper, que é usada para contornar medidas de proteção contra scraping em sites.
2. É importada a biblioteca json para lidar com dados no formato JSON.
3. A variável url é definida como a URL base do site "Status Invest" para opções.
4. São inicializadas três listas vazias: options para armazenar as opções coletadas, assets para armazenar os ativos correspondentes a cada opção, e expirations para armazenar as datas de vencimento das opções.
5. O código entra em um loop for que itera sobre cada ticker na lista de tickers fornecida.
6. Dentro do loop, é criado um objeto scraper usando a função create\_scraper() da biblioteca cloudscraper.
7. É realizada uma requisição GET ao site "Status Invest" com o URL específico para o ticker atual. A resposta é obtida em formato de texto.
8. A biblioteca BeautifulSoup é usada para fazer o parsing do conteúdo HTML da resposta.
9. A variável lists é atribuída à lista de elementos <input> com o atributo name igual a 'options'.
10. Em um loop for, para cada elemento l em lists, são realizadas as seguintes operações:

a. A data de vencimento é extraída do elemento HTML correspondente.

b. O valor do atributo value do elemento l é convertido de JSON para um objeto Python usando a função json.loads(). Esse valor contém as informações das opções em formato de dicionário.

c. As informações das opções são convertidas em um DataFrame do pandas usando pd.DataFrame.from\_records().

d. Uma coluna 'Expiration' é adicionada ao DataFrame opt com a data de vencimento correspondente.

e. O ativo correspondente à primeira opção do DataFrame opt é adicionado à lista assets.

f. O DataFrame opt é adicionado à lista options.

Em caso de exceção (erro) durante o processo de raspagem, é impresso o ticker correspondente.

No final do loop, a variável options é impressa para exibir os dados coletados.

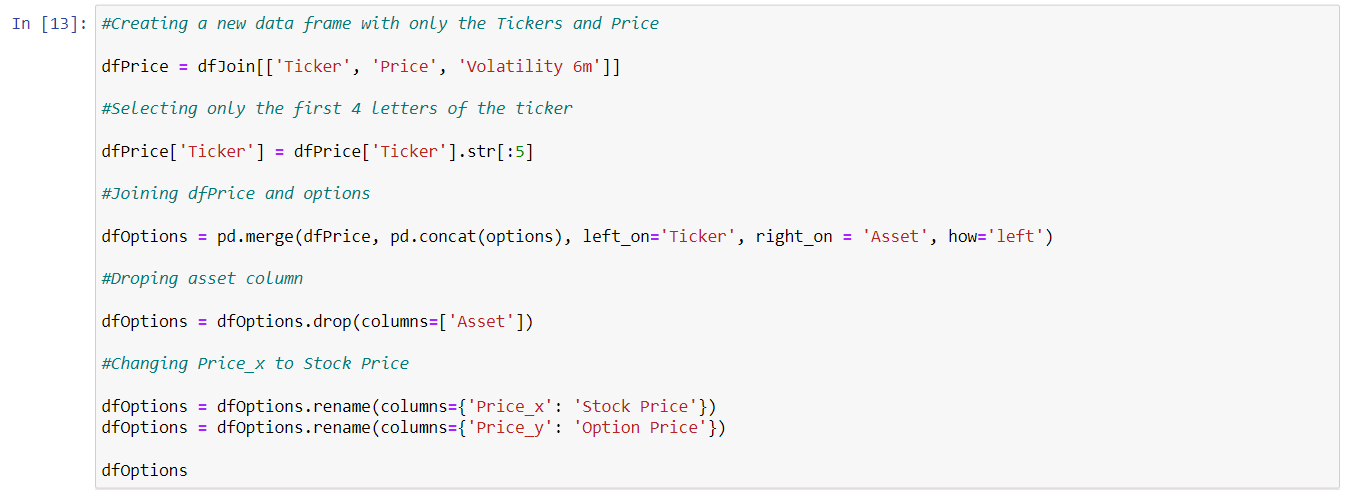
O resultado final será uma lista options contendo DataFrames com as informações das opções coletadas, cada DataFrame correspondendo a um ativo específico, e a lista assets contendo os ativos correspondentes a cada opção.



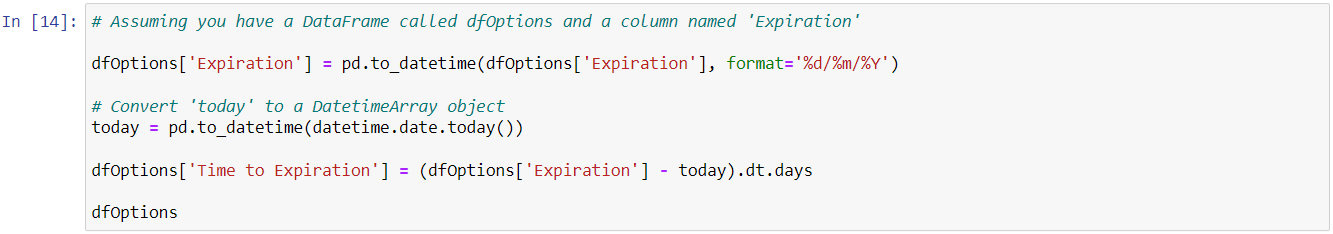
A próxima sequência de código manipula um DataFrame chamado dfJoin para criar um novo DataFrame chamado dfOptions que contém informações sobre os preços das ações e opções de várias empresas.

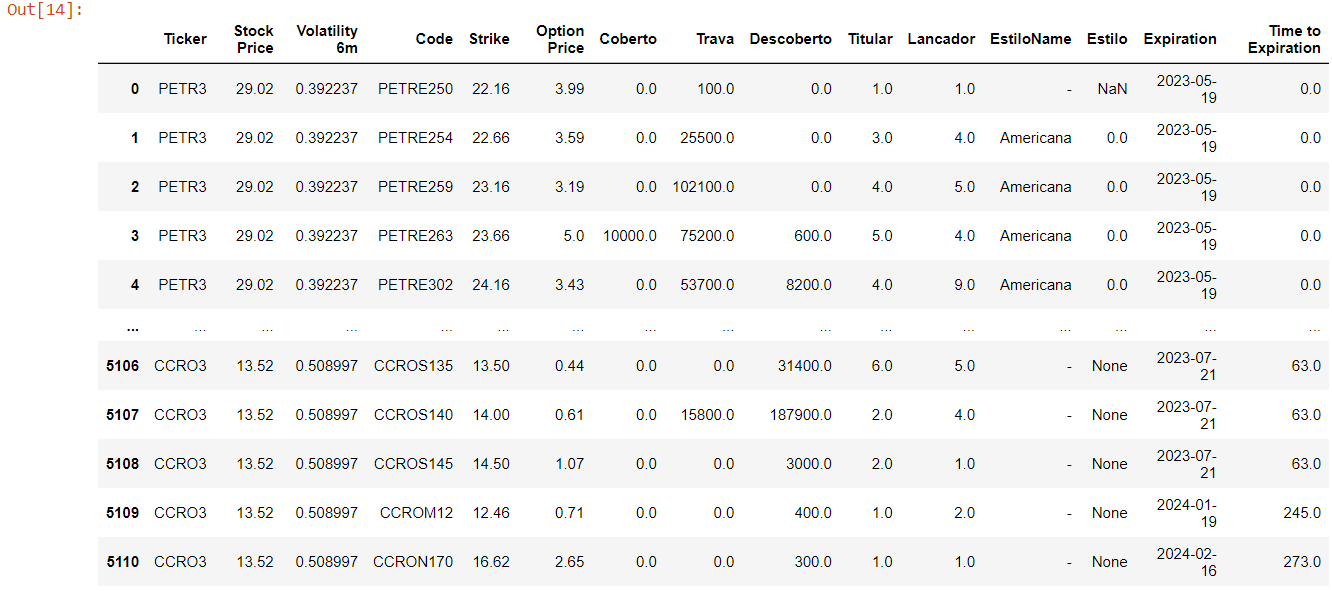
1. O DataFrame dfPrice é criado selecionando apenas as colunas 'Ticker', 'Price' e 'Volatility 6m' do DataFrame dfJoin.
2. A coluna 'Ticker' do DataFrame dfPrice é modificada para conter apenas os quatro primeiros caracteres do ticker, usando a função str[:5].
3. Os DataFrames dfPrice e pd.concat(options) são mesclados com base na coluna 'Ticker' do DataFrame dfPrice e na coluna 'Asset' do DataFrame resultante da concatenação de todos os DataFrames da lista options. A mesclagem é feita com um tipo de junção 'left', preservando todas as linhas do DataFrame dfPrice.
4. A coluna 'Asset' é descartada do DataFrame dfOptions usando a função **drop(columns=['Asset']).**
5. A coluna 'Price\_x' do DataFrame dfOptions é renomeada para 'Stock Price' usando a função **rename(columns={'Price\_x': 'Stock Price'}).**
6. A coluna 'Price\_y' do DataFrame dfOptions é renomeada para 'Option Price' usando a função **rename(columns={'Price\_y': 'Option Price'})**.

Após a execução dessas etapas, o DataFrame dfOptions contém informações sobre os preços das ações ('Stock Price') e os preços das opções ('Option Price') para cada empresa, juntamente com a volatilidade histórica de 6 meses ('Volatility 6m'). Esse DataFrame é utilizado posteriormente em outras análises e cálculos relacionados às opções financeiras.



Por fim, ajustando as datas para “datetime”:





Logo, construímos inúmeras funções para que – no fim – possamos descobrir a probabilidade de default da empresa, portanto:

**1 – BSM\_call**: A função BSM\_call é uma implementação em Python do modelo Black-Scholes-Merton para calcular o preço de uma opção de compra (call option) europeia. A função calcula o preço da opção de compra (call) usando a fórmula do modelo Black-Scholes-Merton. Primeiro, ela calcula dois valores intermediários, d1 e d2, que são utilizados nos cálculos posteriores. Em seguida, utiliza a função de distribuição acumulada normal (CDF) para calcular as probabilidades de d1 e d2 e, finalmente, calcula o preço da opção de compra

**2 – vega**: A função vega calcula a sensibilidade do preço de uma opção em relação à volatilidade do ativo subjacente, conhecida como "vega". Ela é uma medida do impacto que uma variação na volatilidade tem sobre o preço da opção.

**3 – newtons\_method**: A função newtons\_method implementa o método de Newton para estimar a volatilidade implícita de uma opção de compra (call option) com base no preço da opção de compra no mercado. O método de Newton é um algoritmo iterativo usado para encontrar raízes de uma equação não linear. Esse processo é repetido para o número especificado de iterações, refinando gradualmente a estimativa da volatilidade implícita até que o preço estimado da opção se aproxime do preço observado no mercado. No final, a função retorna a estimativa final da volatilidade implícita, que é calculada após as iterações do método de Newton.



**4 – implied\_vol**: A função implied\_vol calcula a volatilidade implícita para cada opção de compra em um DataFrame df usando o método de Newton implementado na função newtons\_method. A volatilidade implícita é a volatilidade do ativo subjacente que é implícita pelo preço da opção de compra no mercado.

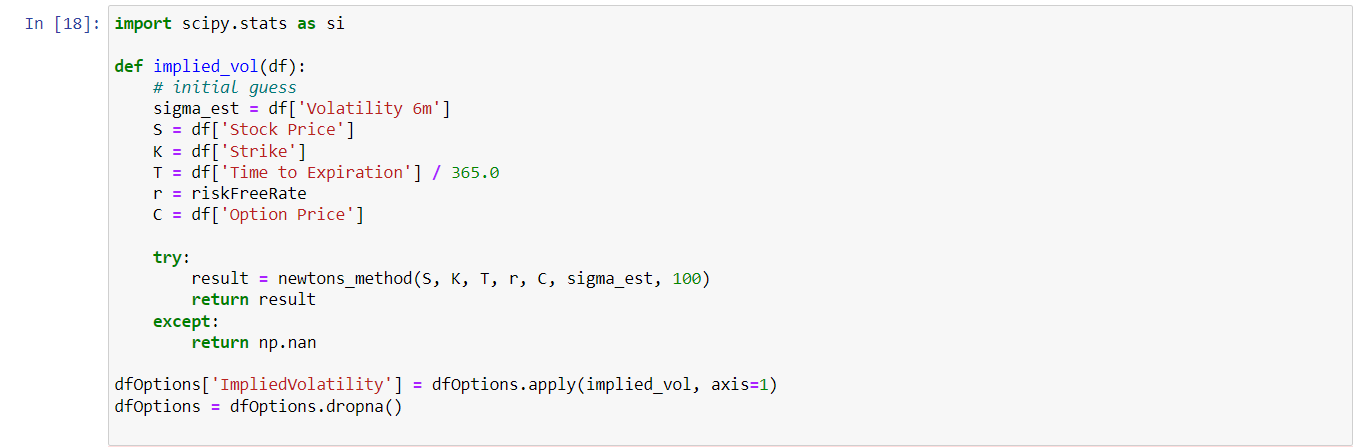
A função recebe o DataFrame df como entrada, que contém informações relevantes para o cálculo da volatilidade implícita de cada opção de compra. O DataFrame deve conter as colunas necessárias, como 'Volatility 6m' (volatilidade histórica de 6 meses), 'Stock Price' (preço atual da ação subjacente), 'Strike' (preço de exercício da opção), 'Time to Expiration' (tempo até a expiração da opção) e 'Option Price' (preço da opção de compra).

A função inicializa a estimativa inicial da volatilidade (sigma\_est) com o valor da coluna 'Volatility 6m' do DataFrame. Em seguida, atribui os valores das outras colunas relevantes a variáveis correspondentes, como S (preço atual da ação), K (preço de exercício), T (tempo até a expiração) e C (preço da opção de compra).

Em seguida, a função chama newtons\_method com os parâmetros adequados, incluindo as variáveis definidas anteriormente, para calcular a volatilidade implícita usando o método de Newton. O número máximo de iterações é definido como 100. Se o cálculo da volatilidade implícita for bem-sucedido (ou seja, não houver erros), a função retorna o resultado do cálculo. Caso contrário, se ocorrer algum erro durante o cálculo (por exemplo, se não for possível encontrar uma solução adequada), a função retorna np.nan para indicar um valor inválido ou ausente.

Posteriormente, a função implied\_vol é aplicada a cada linha do DataFrame dfOptions usando o método apply e atribui os resultados à coluna 'ImpliedVolatility'. Em seguida, as linhas que contêm valores ausentes (NaN) são removidas do DataFrame dfOptions usando dropna().

No final, o DataFrame dfOptions contém a coluna 'ImpliedVolatility', onde cada linha corresponde à volatilidade implícita estimada para a opção de compra correspondente naquela linha.



**5 – d1:** A função d1 calcula o valor de d1 para cada opção de compra em um DataFrame df com base nos parâmetros e fórmula do modelo Black-Scholes-Merton.

A função recebe o DataFrame df como entrada, que contém as informações necessárias para o cálculo de d1 para cada opção de compra. O DataFrame deve conter as colunas relevantes, como 'Volatility 6m' (volatilidade histórica de 6 meses), 'Stock Price' (preço atual da ação subjacente), 'Strike' (preço de exercício da opção), 'Time to Expiration' (tempo até a expiração da opção) e 'Option Price' (preço da opção de compra).

A função extrai os valores das colunas relevantes do DataFrame, como sigma\_est (estimativa inicial da volatilidade), S (preço atual da ação), K (preço de exercício), T (tempo até a expiração) e C (preço da opção de compra). Além disso, a função usa o valor da variável riskFreeRate para a taxa de juros livre de risco.

Em seguida, a função calcula o valor de d1 usando a fórmula do modelo Black-Scholes-Merton:

**d1 = (np.log(S / K) + (r + 0.5 \* sigma\_est \*\* 2) \* T) / (sigma\_est \* np.sqrt(T))**

Onde: np.log é a função logaritmo natural (base e); np.sqrt é a função raiz quadrada.

O valor de d1 representa um dos parâmetros usados na fórmula de precificação de opções. Ele é calculado com base no preço atual da ação, no preço de exercício, no tempo até a expiração, na taxa de juros livre de risco e na volatilidade estimada.

Em seguida, a função retorna o valor de d1 para cada opção de compra. Posteriormente, a função d1 é aplicada a cada linha do DataFrame dfOptions usando o método apply e os resultados são atribuídos à coluna 'Default'. Em seguida, é aplicada a função de distribuição acumulada normal (norm.cdf) aos valores da coluna 'Default', transformando-os em probabilidades. Esses valores transformados são armazenados novamente na coluna 'Default' do DataFrame dfOptions.

No final, o DataFrame dfOptions contém a coluna 'Default', onde cada valor representa a probabilidade associada a d1 para a opção de compra correspondente naquela linha.



**6 – black\_scholes:** A função black\_scholes implementa o modelo Black-Scholes para calcular o valor de mercado da dívida de uma empresa com base em seus ativos totais, dívida total, volatilidade, taxa de juros livre de risco e tempo até o vencimento.

A função recebe os seguintes parâmetros como entrada:

* total\_assets: valor total dos ativos da empresa;
* total\_debt: valor total da dívida da empresa;
* volatility: volatilidade dos ativos da empresa;
* risk\_free\_rate: taxa de juros livre de risco;
* time\_to\_maturity: tempo até o vencimento da dívida.

A função calcula o valor de d1 e d2 usando as fórmulas do modelo Black-Scholes: **d1 = (np.log(total\_assets / total\_debt) + (risk\_free\_rate + 0.5 \* volatility \*\* 2) \* time\_to\_maturity) / (volatility \* np.sqrt(time\_to\_maturity))**

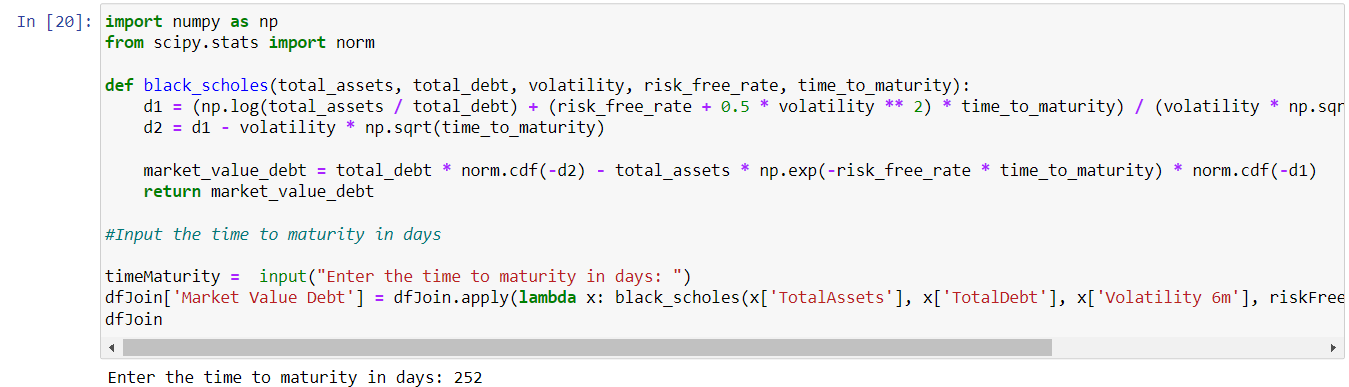
**d2 = d1 - volatility \* np.sqrt(time\_to\_maturity)**

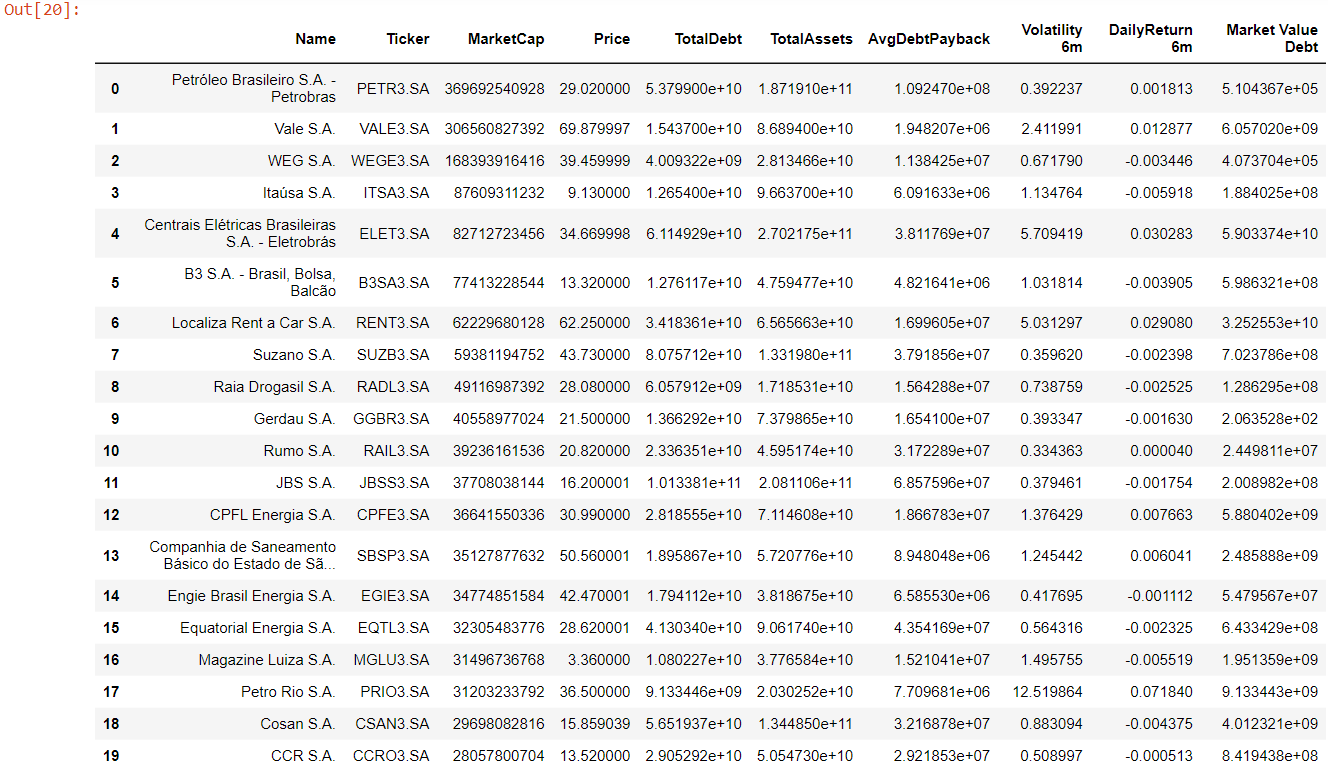
Em seguida, a função calcula o valor de mercado da dívida (market\_value\_debt) usando as fórmulas do modelo Black-Scholes: **market\_value\_debt = total\_debt \* norm.cdf(-d2) - total\_assets \* np.exp(-risk\_free\_rate \* time\_to\_maturity) \* norm.cdf(-d1)**

Onde: norm.cdf é a função de distribuição acumulada normal, que retorna a probabilidade de um valor estar abaixo de um certo limite. np.exp é a função exponencial, que calcula o valor exponencial de um número.

O valor de mercado da dívida é calculado com base no valor total dos ativos, valor total da dívida, volatilidade, taxa de juros livre de risco e tempo até o vencimento. Ele é determinado pela diferença entre o valor presente dos pagamentos de dívida esperados (expressos como uma probabilidade usando norm.cdf) e o valor presente dos fluxos de caixa esperados dos ativos da empresa. No código, a função black\_scholes é aplicada a cada linha do DataFrame dfJoin usando o método apply. Os resultados são atribuídos à coluna 'Market Value Debt' do DataFrame dfJoin. O valor de timeMaturity é solicitado ao usuário como entrada e é convertido para anos dividindo-o por 365 antes de ser passado para a função black\_scholes.

No final, o DataFrame dfJoin contém a coluna 'Market Value Debt', onde cada valor representa o valor de mercado da dívida calculado pela função black\_scholes para a respectiva linha.





**7 – default\_probability:** A função default\_probability calcula a probabilidade de default (falência) de uma empresa com base em seus ativos totais, dívida total, taxa de juros livre de risco, volatilidade dos ativos e tempo até o vencimento.

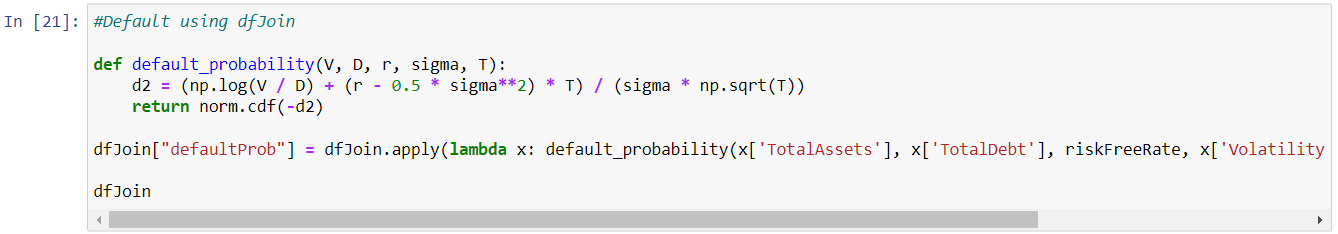
* A função recebe os seguintes parâmetros como entrada:
* V: valor total dos ativos da empresa;
* D: valor total da dívida da empresa;
* r: taxa de juros livre de risco;
* sigma: volatilidade dos ativos da empresa;
* T: tempo até o vencimento da dívida.

A função calcula o valor de d2 usando a fórmula do modelo Black-Scholes: **d2 = (np.log(V / D) + (r - 0.5 \* sigma\*\*2) \* T) / (sigma \* np.sqrt(T))**

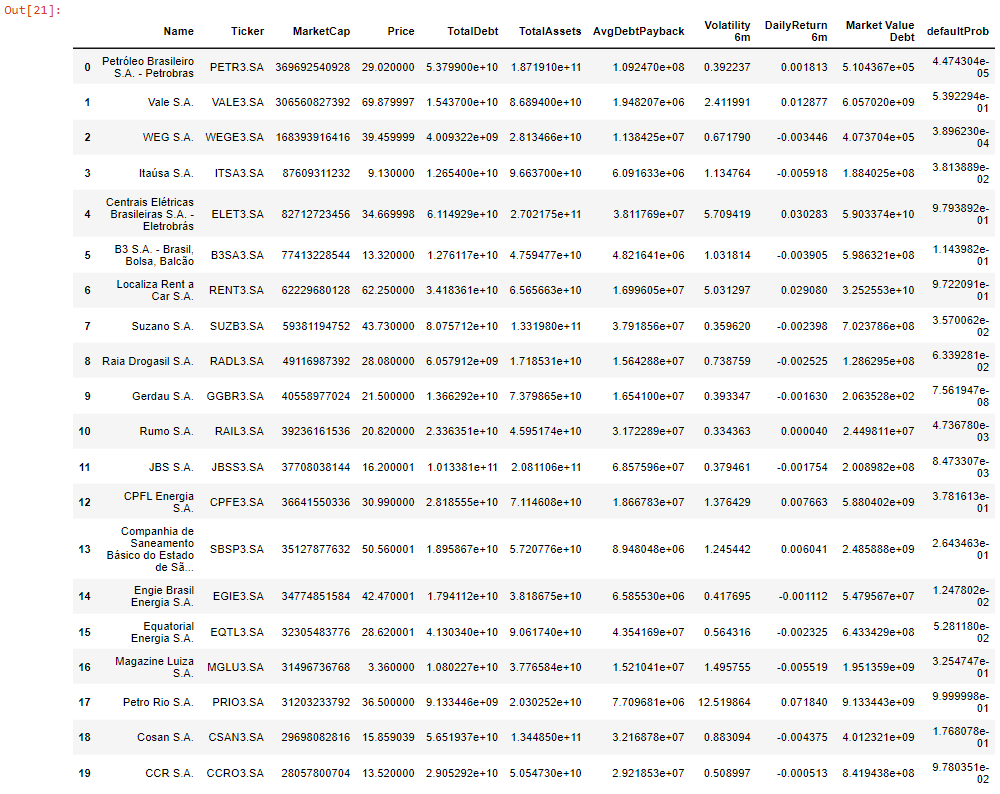
Em seguida, a função retorna a probabilidade de default da empresa usando a função de distribuição acumulada normal (norm.cdf) com o valor -d2. A probabilidade de default é a probabilidade de que o valor dos ativos da empresa seja inferior ao valor da dívida no momento do vencimento.

No código fornecido, a função default\_probability é aplicada a cada linha do DataFrame dfJoin usando o método apply. Os resultados são atribuídos à coluna 'defaultProb' do DataFrame dfJoin. Os parâmetros para a função são extraídos das colunas relevantes do DataFrame dfJoin, como 'TotalAssets', 'TotalDebt', 'Volatility 6m' e 'timeMaturity'.

No final, o DataFrame dfJoin contém a coluna 'defaultProb', onde cada valor representa a probabilidade de default calculada pela função default\_probability para a respectiva linha da empresa. Essa probabilidade indica a chance de a empresa entrar em default com base nos parâmetros fornecidos.



Com resultado para cada uma das empresas:



**REFERÊNCIAS:**

<https://aaronschlegel.me/black-scholes-formula-python.html>

<https://www.tradingview.com/>

<https://www.statusinvest.com.br/>