**Program Design Document (PDD) - scrip-1.py**

**1. Introdução**

Este documento descreve o design e a arquitetura do script Python teste-1.py, que tem como objetivo coletar dados de ações financeiras através de web scraping do site Yahoo Finance. O script foi desenvolvido para ser uma ferramenta autônoma e eficiente para a extração de informações de mercado em tempo real, como preço, variação, volume, capitalização de mercado e relação P/E, e para persistir esses dados em um formato estruturado (CSV).

**1.1. Propósito**

O principal propósito do teste-1.py é fornecer uma solução programática para a coleta automatizada de dados de ações do Yahoo Finance. Isso permite que usuários e sistemas obtenham informações de mercado atualizadas sem a necessidade de interação manual com o site, facilitando análises, monitoramento e integração com outras aplicações.

**1.2. Escopo**

O escopo deste script abrange:

•Coleta de Dados: Extração de dados de cotação de ações (preço, variação, volume, etc.) de páginas individuais do Yahoo Finance.

•Processamento de Dados: Parsing do HTML para identificar e extrair os elementos de dados relevantes.

•Persistência de Dados: Salvamento dos dados coletados em um arquivo CSV.

•Gerenciamento de Requisições: Implementação de atrasos aleatórios para mitigar o risco de bloqueio por parte do servidor.

•Registro de Eventos: Utilização de logging para monitorar o progresso e diagnosticar problemas.

O script não abrange:

•Coleta de dados históricos ou financeiros detalhados (balanços, demonstrações de resultados).

•Autenticação ou interação com formulários no Yahoo Finance.

•Análise de dados avançada ou visualização.

•Integração com bancos de dados ou outros sistemas de armazenamento além de CSV.

1.3. Definições, Acrônimos e Abreviações

•Web Scraping: Técnica de extração de dados de websites.

•HTML: HyperText Markup Language, a linguagem padrão para criação de páginas web.

•BeautifulSoup: Biblioteca Python para parsing de documentos HTML e XML.

•Pandas: Biblioteca Python para manipulação e análise de dados, fornecendo estruturas de dados como DataFrames.

•CSV: Comma Separated Values, um formato de arquivo de texto simples para armazenar dados tabulares.

•HTTP: Hypertext Transfer Protocol, o protocolo para transferência de informações na World Wide Web.

•User-Agent: Um cabeçalho HTTP que identifica o aplicativo, sistema operacional, fornecedor ou versão do agente do usuário que fez a requisição.

•Rate Limiting: Mecanismo usado por servidores para controlar o número de requisições que um cliente pode fazer em um determinado período.

•PDD: Program Design Document.

**2. Design de Alto Nível**

O teste-1.py segue um design modular e orientado a objetos, encapsulando a lógica de web scraping em uma classe dedicada. Isso promove a reutilização de código, a manutenibilidade e a clareza da estrutura.

**2.1. Arquitetura Geral**

A arquitetura do script é linear e sequencial, com uma classe central (YahooFinanceScraper) que orquestra as operações de coleta. A função main() atua como o ponto de entrada, inicializando o scraper, definindo as ações a serem coletadas e chamando os métodos apropriados para executar o scraping e salvar os dados.

**2.2. Componentes Principais**

•YahooFinanceScraper Classe: O coração do script, responsável por toda a lógica de web scraping.

•Métodos Auxiliares Privados (\_ prefixo): Funções internas da classe que lidam com detalhes de implementação, como requisições HTTP (\_fetch\_page), parsing de HTML (\_parse\_data) e geração de atrasos (\_get\_random\_delay).

•Métodos Públicos: Funções da classe que expõem a funcionalidade principal, como scrape\_stock, scrape\_multiple\_stocks e save\_to\_csv.

•main() Função: O ponto de entrada do script, responsável pela configuração e execução do fluxo principal.

•Módulo logging: Para fornecer feedback sobre o progresso e erros durante a execução.

**3. Design Detalhado**

**3.1. Classe YahooFinanceScraper**

**3.1.1. Atributos**

•output\_dir (str): Caminho para o diretório onde os arquivos CSV serão salvos. Inicializado no construtor.

•base\_url (str): URL base do Yahoo Finance para cotações de ações (https://finance.yahoo.com/quote/).

•headers (dict): Dicionário de cabeçalhos HTTP para simular uma requisição de navegador, incluindo User-Agent, Accept, etc.

•session (requests.Session): Objeto de sessão requests para persistir parâmetros entre requisições (ex: cabeçalhos).

**3.1.2. Métodos**

•\*\*`init(self, output\_dir: str =

data"): - \*\*Propósito\*\*: Inicializar a instância do scraper, configurar o diretório de saída e os cabeçalhos HTTP. - \*\*Lógica\*\*: Cria o diretório output\_dirse não existir. Configura a sessãorequests` com os cabeçalhos definidos.

•\_get\_random\_delay(self) -> float:

•Propósito: Gerar um atraso aleatório para evitar detecção e bloqueio por parte do servidor.

•Lógica: Retorna um número float aleatório entre 1.0 e 3.0 segundos usando random.uniform().

•\_fetch\_page(self, symbol: str) -> Optional[str]:

•Propósito: Realizar a requisição HTTP para a página de cotação de uma ação específica no Yahoo Finance.

•Lógica: Constrói a URL da ação. Utiliza self.session.get() para fazer a requisição. Verifica o status\_code da resposta. Em caso de sucesso (200), retorna o conteúdo HTML. Em caso de falha, registra um aviso ou erro e retorna None.

•Tratamento de Erros: Captura requests.exceptions.RequestException para lidar com problemas de rede ou timeout.

•\_parse\_data(self, html: str, symbol: str) -> Optional[Dict]:

•Propósito: Extrair os dados relevantes da ação a partir do HTML da página.

•Lógica: Utiliza BeautifulSoup para analisar o HTML. Localiza elementos específicos usando find() com base em tags, classes e atributos (data-field, data-test). Extrai o texto desses elementos. Mapeia os dados extraídos para um dicionário com chaves padronizadas. Adiciona a data/hora da coleta e a fonte.

•Tratamento de Erros: Captura exceções gerais durante o parsing para lidar com estruturas HTML inesperadas e registra erros.

•scrape\_stock(self, symbol: str) -> Optional[Dict]:

•Propósito: Orquestrar a coleta de dados para uma única ação.

•Lógica: Chama \_fetch\_page para obter o HTML. Se bem-sucedido, chama \_parse\_data para extrair os dados. Retorna os dados processados ou None.

•scrape\_multiple\_stocks(self, symbols: List[str]) -> pd.DataFrame:

•Propósito: Coletar dados para uma lista de ações.

•Lógica: Itera sobre a lista de symbols. Para cada símbolo, chama scrape\_stock. Adiciona um atraso aleatório (time.sleep(self.\_get\_random\_delay())) após cada requisição para evitar bloqueios. Constrói um pd.DataFrame a partir dos dados coletados.

•save\_to\_csv(self, df: pd.DataFrame, filename: str = None) -> str:

•Propósito: Salvar o DataFrame de dados em um arquivo CSV.

•Lógica: Se filename não for fornecido, gera um nome de arquivo único baseado na data e hora atuais. Constrói o caminho completo do arquivo usando os.path.join(). Utiliza df.to\_csv() para salvar o DataFrame, sem incluir o índice.

**3.2. Função main()**

•Propósito: Ponto de entrada e controle do fluxo de execução do script.

•Lógica: Define uma lista padrão de símbolos de ações. Instancia YahooFinanceScraper. Chama scrape\_multiple\_stocks para iniciar a coleta. Se dados forem retornados, chama save\_to\_csv e imprime os dados no console. Caso contrário, imprime uma mensagem de falha.

**4. Requisitos Não Funcionais**

•Performance: O script é projetado para coletar dados de um número limitado de ações em tempo real. A performance é impactada pelos atrasos intencionais para evitar bloqueios. Não é otimizado para coleta em massa de milhares de ações simultaneamente.

•Confiabilidade: A confiabilidade da coleta de dados depende da estabilidade da estrutura HTML do Yahoo Finance e da capacidade de evitar bloqueios. Erros de rede e parsing são tratados para evitar falhas completas.

•Segurança: O script não lida com informações sensíveis. No entanto, o uso de User-Agent e atrasos aleatórios são medidas de segurança para o próprio scraper, protegendo-o de ser banido.

•Manutenibilidade: O código é modular e bem comentado, facilitando a compreensão e futuras modificações. A separação de responsabilidades entre os métodos da classe contribui para a manutenibilidade.

•Escalabilidade: A escalabilidade é limitada pela natureza do web scraping e pelos limites de taxa do Yahoo Finance. Para maior escalabilidade, seria necessário o uso de proxies rotativos, CAPTCHA solvers ou uma API oficial (se disponível).

**5. Plano de Testes (Visão Geral)**

Os testes para este script devem focar na validação da coleta de dados e na resiliência a falhas de rede ou alterações no site.

•Testes Unitários: Embora não implementados no script atual, testes unitários seriam ideais para:

•\_get\_random\_delay(): Verificar se retorna um float dentro do intervalo esperado.

•\_fetch\_page(): Testar com URLs válidas e inválidas, simular erros de rede e diferentes códigos de status HTTP.

•\_parse\_data(): Testar com diferentes estruturas HTML (válidas e inválidas) para garantir que os dados sejam extraídos corretamente ou que o tratamento de erros funcione.

•save\_to\_csv(): Verificar se o arquivo CSV é criado corretamente e se o conteúdo corresponde ao DataFrame.

•Testes de Integração: Testar o fluxo completo:

•Executar o script com uma lista de símbolos de ações conhecidos e verificar se o arquivo CSV é gerado com os dados esperados.

•Testar com símbolos de ações inexistentes para verificar o tratamento de erros.

•Monitorar o comportamento do script ao longo do tempo para identificar possíveis bloqueios ou alterações no site.

•Testes de Performance: Medir o tempo de execução para diferentes quantidades de ações para entender as limitações de taxa e o impacto dos atrasos.

**6. Considerações de Implantação**

O script é autônomo e pode ser executado em qualquer ambiente Python 3.x com as dependências instaladas. Não há requisitos de infraestrutura complexos.

•Ambiente: Python 3.x.

•Dependências: requests, beautifulsoup4, pandas.

•Execução: Via linha de comando (python teste-1.py).

•Saída: Arquivos CSV gerados no diretório data/ (ou configurado).

7. Melhorias e Evolução Futura

As melhorias sugeridas na documentação (documentation\_teste-1.md) também se aplicam ao design e podem ser consideradas para futuras iterações:

•Configuração Externa: Implementar um arquivo de configuração para variáveis como symbols e output\_dir.

•Tratamento de Erros Aprimorado: Adicionar retries com backoff exponencial para requisições HTTP.

•Paralelização: Explorar o uso de threading ou asyncio para coletar dados de múltiplas ações de forma mais eficiente, mantendo o respeito aos limites de taxa.

•Persistência em Banco de Dados: Integrar com um banco de dados (SQLite, PostgreSQL, MongoDB) para armazenamento mais robusto e consultas complexas.

•Dockerização: Criar um Dockerfile para empacotar o script e suas dependências, garantindo um ambiente de execução consistente e facilitando a implantação.

8. Autor

Lauro Bonometti.