Projeto de Geração de Relatórios de Análise de Concorrentes do Instagram com IA

Parte I: Blueprint Arquitetural para uma Ferramenta de Análise com IA

Esta parte estabelece os princípios fundamentais e a estrutura do projeto, enfatizando as melhores práticas para a criação de uma aplicação manutenível, escalável e segura, adequada para um portfólio público no GitHub.

Seção 1.1: Estruturando o Projeto para Escalabilidade e Colaboração

A base de qualquer projeto de software robusto e profissional não reside apenas na funcionalidade do seu código, mas também na clareza e lógica da sua organização. Para um projeto destinado a um portfólio no GitHub, uma estrutura de diretórios bem definida é fundamental, pois serve como um mapa para qualquer pessoa que interaja com o código, incluindo o próprio desenvolvedor no futuro.[1] Uma estrutura que separa claramente as responsabilidades — dados, código-fonte, notebooks experimentais, configurações e relatórios gerados — torna o projeto mais intuitivo, fácil de manter e preparado para futuras expansões.

A arquitetura de diretórios adotada para este projeto é baseada em convenções estabelecidas na comunidade de ciência de dados e engenharia de software, garantindo que cada componente tenha seu lugar designado.[1]

- **README.md**: Este arquivo é o ponto de entrada principal do repositório. Ele deve articular de forma concisa o propósito do projeto (o "porquê"), sua funcionalidade (o "o quê") e como configurá-lo e executá-lo (o "como"). A

clareza neste documento é essencial para fornecer o contexto necessário sem a necessidade de explicações verbais repetitivas.[1]

- **main.py**: O ponto de entrada da aplicação. Este script orquestra todo o fluxo de trabalho: desde a leitura do briefing inicial, passando pela ingestão e análise de dados, até a geração do relatório final.
- **requirements.txt**: Um arquivo de texto que lista todas as dependências do projeto. Isso garante que o ambiente possa ser replicado de forma consistente em qualquer máquina.[3]
- **.env**: Um arquivo para armazenar credenciais sensíveis, como tokens de API. Este arquivo é fundamental para a segurança e nunca deve ser submetido ao controle de versão.[5]
- **.gitignore**: Um arquivo de configuração para o Git que especifica quais arquivos e diretórios devem ser ignorados pelo controle de versão. Essencial para excluir o arquivo .env, os ambientes virtuais, os caches do Python e os diretórios de dados e relatórios.[4]
- **config/**: Um módulo Python para configurações e constantes de toda a aplicação, como nomes de arquivos, caminhos de diretórios e parâmetros de análise.
- **data/**: Um diretório para armazenar os dados coletados e processados.
  - raw/: Armazena os dados brutos em formato JSON, exatamente como foram recebidos da API da Apify.
  - processed/: Contém os dados após a limpeza e transformação, geralmente em formatos como dataframes do pandas salvos em CSV ou Parquet.
  - É crucial que o diretório data/ seja incluído no .gitignore para evitar o commit de grandes volumes de dados no repositório.
- **src/**: O diretório principal do código-fonte, estruturado como um pacote Python para permitir importações modulares.
  - __init__.py: Um arquivo vazio que sinaliza ao Python que o diretório src é um pacote.
  - data_ingestion/: Módulo responsável por todas as interações com a API da Apify, incluindo a execução de "Actors" e o tratamento de erros.
  - analysis/: Módulo que contém toda a lógica de análise, tanto quantitativa (cálculo de KPIs) quanto qualitativa (análise de conteúdo com LLMs).
  - reporting/: Módulo dedicado à geração do relatório final em formato .docx, incluindo a criação de tabelas, gráficos e a aplicação de estilos.
- **notebooks/**: Um espaço para análise exploratória e prototipagem usando Jupyter Notebooks. Manter os notebooks separados do código de produção da

aplicação principal é uma prática recomendada para manter a clareza e a organização.[1]

- **reports/**: O diretório de saída onde os relatórios .docx gerados serão salvos. Este diretório também deve ser ignorado pelo Git.
- **templates/**: Este diretório armazena o documento Word (template.docx) que serve como modelo de estilo para os relatórios. A utilização de um template é uma decisão arquitetural chave para contornar as limitações de estilização programática da biblioteca python-docx.[7]

A tabela a seguir visualiza essa estrutura, servindo como um guia de referência rápida para a organização do projeto.

Tabela 1: Estrutura de Diretórios do Projeto

Caminho do Diretório/Arquivo	Descrição
project/	Diretório raiz do projeto.
├── README.md	Documentação principal com visão geral, instalação e uso.
├── main.py	Script principal que orquestra a execução da aplicação.
├── requirements.txt	Lista de dependências Python para reprodutibilidade do ambiente.
├── .env	Arquivo para armazenamento seguro de chaves de API e outras variáveis de ambiente.
├── .gitignore	Configuração para excluir arquivos e diretórios do controle de versão.
├── config/	Módulo para configurações globais da aplicação.
├── data/	Diretório para dados (ignorado pelo Git).
│ ├── raw/	Armazena os dados brutos JSON da API.
│ └── processed/	Armazena dados limpos e processados.

├── src/	Diretório do código-fonte, estruturado como um pacote.
│ ├── __init__.py	Torna src um pacote Python.
│ ├── data_ingestion/	Módulo para extração de dados da Apify.
│ ├── analysis/	Módulo para análise de dados e geração de insights.
│ └── reporting/	Módulo para a criação do relatório .docx.
├── notebooks/	Para notebooks de análise exploratória e prototipagem.
├── reports/	Diretório de saída para os relatórios gerados (ignorado pelo Git).
└── templates/	Contém o template.docx com estilos pré-definidos.

A adoção dessa estrutura não é um mero formalismo; é uma decisão estratégica. Ela transforma o que poderia ser um simples script em uma aplicação bem-arquitetada, demonstrando uma compreensão de práticas de engenharia de software que são valorizadas em qualquer contexto profissional.

Seção 1.2: Gerenciamento de Dependências e Ambiente

Um ambiente de desenvolvimento limpo, isolado e reprodutível é a pedra angular de um projeto Python profissional. Sem um gerenciamento de dependências adequado, surgem problemas de conflitos de versões, e a colaboração torna-se impraticável. Esta seção detalha as etapas críticas para configurar o ambiente do projeto de forma robusta e segura.

Ambientes Virtuais (venv)
A prática mais fundamental no desenvolvimento Python é o uso de ambientes virtuais. Um ambiente virtual é um diretório autocontido que contém uma instalação específica do Python e pacotes adicionais, isolando o projeto das dependências de outros projetos e da instalação global do sistema.2 Isso previne o "inferno de dependências", onde diferentes projetos podem requerer versões conflitantes da mesma biblioteca. A criação de um ambiente virtual é simples e deve ser o primeiro passo antes da instalação de qualquer pacote:

Bash

```
# Navegue até o diretório raiz do projeto
python -m venv venv

# Ative o ambiente virtual
# No Windows:
venv\Scripts\activate
# No macOS/Linux:
source venv/bin/activate
```

Arquivo de Dependências requirements.txt
Uma vez que o ambiente virtual está ativo, todas as bibliotecas instaladas com pip serão contidas nele. Para garantir que outros colaboradores (ou o próprio desenvolvedor em uma nova máquina) possam recriar este ambiente com precisão, as dependências são listadas em um arquivo requirements.txt.4 Este arquivo é um simples manifesto de pacotes e suas versões.
As bibliotecas essenciais para este projeto incluem:

- apify-client: Para interagir com a plataforma Apify.
- langchain e suas dependências (como langchain-openai): Para orquestrar as interações com o Large Language Model (LLM).
- python-docx: Para criar e manipular programaticamente os documentos Word (.docx).
- pandas: Para a manipulação e análise de dados tabulares.
- matplotlib e seaborn: Para a criação de visualizações de dados.
- python-dotenv: Para carregar variáveis de ambiente de um arquivo .env.

Após instalar os pacotes necessários (pip install pandas matplotlib seaborn...), o arquivo requirements.txt pode ser gerado automaticamente com o seguinte comando, que "congela" o estado atual dos pacotes instalados no ambiente [3]:

Bash

```
pip freeze > requirements.txt
```

Para instalar as dependências em um novo ambiente, o comando é igualmente simples:

```
Bash
```

```
pip install -r requirements.txt
```

Gerenciamento Seguro de Credenciais com .env
A segurança é uma preocupação primordial em qualquer aplicação que utilize APIs externas. Chaves de API e tokens de acesso são segredos que nunca devem ser codificados diretamente no código-fonte ou, pior ainda, submetidos a um repositório Git público. A prática padrão da indústria é armazenar essas credenciais como variáveis de ambiente.5
A biblioteca python-dotenv simplifica esse processo durante o desenvolvimento, permitindo que as variáveis de ambiente sejam carregadas a partir de um arquivo de texto chamado .env localizado no diretório raiz do projeto.[5]

1. **Crie o arquivo .env**: Na raiz do projeto, crie um arquivo com o nome .env.
2. **Adicione as chaves**: Dentro do arquivo .env, adicione as chaves de API necessárias no formato CHAVE=VALOR:
   ```
   #.env
   APIFY_API_TOKEN="apify_api_..."
   OPENAI_API_KEY="sk-..."
   ```
3. **Ignore o arquivo no Git**: Adicione a linha .env ao seu arquivo .gitignore. Esta é a etapa de segurança mais crítica, pois impede que o Git rastreie e envie seus segredos para o repositório.
4. **Carregue as variáveis no código**: No início da sua aplicação (por exemplo, em main.py), use a biblioteca python-dotenv para carregar essas variáveis no ambiente de execução do Python:
   ```
   Python
   import os
   from dotenv import load_dotenv

   # Carrega as variáveis do arquivo.env para o ambiente
   load_dotenv()

   # Agora, as variáveis podem ser acessadas usando os.getenv()
   ```

```
apify_token = os.getenv("APIFY_API_TOKEN")
openai_key = os.getenv("OPENAI_API_KEY")
```

A escolha de usar requirements.txt em vez de ferramentas mais avançadas como Poetry ou Pipenv [4] é uma decisão deliberada. Embora essas ferramentas ofereçam uma resolução de dependências mais sofisticada,

requirements.txt é mais simples, universalmente compreendido e perfeitamente adequado para um projeto de portfólio. Ele demonstra competência no gerenciamento de dependências sem introduzir uma camada de complexidade que poderia desviar o foco da funcionalidade principal da aplicação: a análise do Instagram.

Parte II: O Motor de Dados - Automatizando a Inteligência do Instagram com a Apify

Esta parte detalha o pipeline de aquisição de dados, desde a seleção criteriosa da ferramenta na plataforma Apify até a implementação robusta do cliente e o tratamento dos erros inerentes ao processo de web scraping.

Seção 2.1: Selecionando o Scraper de Instagram Ideal

A escolha da ferramenta de extração de dados é uma decisão arquitetural fundamental que impacta diretamente a qualidade e a profundidade da análise subsequente. As APIs oficiais do Instagram (Basic Display e Graph API) são excessivamente restritivas para uma análise de concorrentes abrangente. Elas não permitem a extração de posts de usuários comuns em larga escala, informações detalhadas de perfis ou comentários, tornando-as inadequadas para este caso de uso.[10] Isso nos leva à necessidade de utilizar ferramentas de web scraping de terceiros, como os "Actors" disponíveis na Apify Store.

Uma análise comparativa dos Actors disponíveis revela diferentes especializações e modelos de precificação:

- **apify/instagram-profile-scraper**: Esta ferramenta é otimizada para extrair dados agregados de perfis. Fornece métricas essenciais como número de seguidores e de contas seguidas, biografia, contagem de posts e até perfis relacionados.[11] Sua saída é ideal para construir um painel de visão geral dos concorrentes.

- **apify/instagram-post-scraper**: Focado na extração de dados em nível de post individual. Este Actor é indispensável para uma análise de conteúdo profunda, pois fornece legendas, contagens de curtidas e comentários, hashtags, menções e URLs de mídia.[12] A estrutura de saída JSON é rica e bem definida, facilitando o processamento posterior.[12]
- **apify/instagram-hashtag-scraper**: Projetado para pesquisar conteúdo com base em hashtags específicas.[13] Embora seja poderoso para pesquisa de tendências e tópicos, não é a ferramenta principal para uma comparação direta entre perfis de concorrentes. Pode ser considerado para uma extensão futura do projeto.
- **apidojo/instagram-scraper**: Um Actor versátil, descrito como um "canivete suíço", capaz de extrair uma vasta gama de dados.[15] No entanto, seu modelo de precificação "pay-per-result" (pagamento por resultado) pode ser menos previsível e potencialmente mais caro para o nosso caso de uso, que envolve a extração de um número definido de posts de um conjunto fixo de concorrentes.

Decisão Estratégica: A abordagem mais eficaz e com melhor custo-benefício para este projeto é uma combinação de ferramentas. Utilizaremos o **apify/instagram-profile-scraper** para obter as estatísticas de resumo de cada concorrente e o **apify/instagram-post-scraper** para realizar a análise detalhada do conteúdo de seus posts recentes. Essa estratégia de duas frentes garante um conjunto de dados abrangente, cobrindo tanto a performance geral do perfil quanto a eficácia do conteúdo específico.

A tabela a seguir resume a análise e justifica a seleção das ferramentas.

Tabela 2: Comparação de Actors da Apify para Instagram

Nome do Actor	Caso de Uso Principal	Pontos de Dados Chave Fornecidos	Modelo de Precificação	Selecionado (S/N)
apify/instagram-profile-scraper	Análise de métricas de perfil	Seguidores, seguindo, biografia, contagem de posts, taxa de engajamento	Por execução/uso de plataforma	S
apify/instagram-	Análise de	Legendas,	Por	S

post-scraper	conteúdo detalhada	curtidas, comentários, hashtags, tipo de mídia, URLs	execução/uso de plataforma	
apify/instagram-hashtag-scraper	Pesquisa de tendências e tópicos	Posts associados a uma hashtag, dados do post	Por resultado ($2.30 / 1.000 resultados) [13]	N
apidojo/instagram-scraper	Extração de dados diversificada	Posts, perfis, localizações, reels	Por resultado ($0.50 / 1.000 posts) [15]	N

Formalizar essa decisão em uma tabela não apenas documenta o processo de pensamento, mas também demonstra uma abordagem analítica e metódica para a seleção de ferramentas — uma característica de um trabalho de nível sênior.

Seção 2.2: Implementando o Cliente Python da Apify

Com as ferramentas selecionadas, o próximo passo é interagir com a plataforma Apify de forma programática usando a biblioteca apify-client. Este processo envolve inicializar o cliente, executar os Actors com os parâmetros corretos e recuperar os dados resultantes.

1. Inicialização do Cliente
O primeiro passo é instanciar o cliente da Apify, fornecendo o token de API que foi previamente armazenado no arquivo .env.

Python

```
# Em src/data_ingestion/apify_handler.py
import os
from apify_client import ApifyClient
```

```
def get_apify_client():
    """Inicializa e retorna uma instância do cliente da Apify."""
    apify_token = os.getenv("APIFY_API_TOKEN")
    if not apify_token:
        raise ValueError("Token da API da Apify não encontrado. Verifique seu arquivo.env.")
    return ApifyClient(apify_token)
```

2. Execução de um Actor

A execução de um Actor é feita chamando-o pelo seu nome (ex: apify/instagram-profile-scraper) e passando um dicionário de entrada (run_input) que contém os parâmetros necessários. Para os scrapers de perfil e de posts, a entrada principal será uma lista de nomes de usuário do Instagram a serem analisados.11

O método actor.call() é particularmente conveniente, pois ele inicia a execução, aguarda sua conclusão e retorna o objeto de execução contendo informações sobre os resultados.[16]

Python

```
# Em src/data_ingestion/apify_handler.py

def scrape_profile_data(client: ApifyClient, usernames: list[str]):
    """Executa o Instagram Profile Scraper para uma lista de nomes de usuário."""
    print(f"Iniciando a extração de dados de perfil para: {usernames}")

    actor = client.actor("apify/instagram-profile-scraper")
    run_input = {"usernames": usernames}

    run = actor.call(run_input=run_input)
    print("Extração de dados de perfil concluída.")
    return run

# Função similar para o instagram-post-scraper...
def scrape_post_data(client: ApifyClient, usernames: list[str], max_posts: int = 50):
    """Executa o Instagram Post Scraper."""
    print(f"Iniciando a extração de posts para: {usernames}")

    actor = client.actor("apify/instagram-post-scraper")
```

```
    # O input pode variar entre actors. Consulte a documentação do Actor na Apify.
    run_input = {
        "usernames": usernames,
        "resultsLimit": max_posts
    }

    run = actor.call(run_input=run_input)
    print("Extração de posts concluída.")
    return run
```

3. Recuperação e Armazenamento dos Dados
Após a conclusão da execução, os dados são armazenados em um "Dataset" na plataforma Apify. O cliente pode ser usado para buscar esses dados. O objeto run retornado pelo método call() contém o ID do dataset padrão (defaultDatasetId), que pode ser usado para acessar os resultados.16 Os dados são então salvos localmente em um arquivo JSON no diretório
data/raw/ para persistência e processamento futuro.

Python

```
# Em src/data_ingestion/apify_handler.py
import json

def get_data_from_run(client: ApifyClient, run: dict, output_path: str):
    """Recupera os itens de um dataset de uma execução e salva em um arquivo JSON."""
    dataset_id = run.get("defaultDatasetId")
    if not dataset_id:
        print("Nenhum dataset encontrado na execução.")
        return

    print(f"Recuperando dados do dataset: {dataset_id}")
    dataset_items = client.dataset(dataset_id).list_items().items

    with open(output_path, 'w', encoding='utf-8') as f:
        json.dump(dataset_items, f, ensure_ascii=False, indent=4)
```

```
    print(f"Dados salvos em: {output_path}")
    return dataset_items
```

Este fluxo de trabalho — inicializar, executar, aguardar e recuperar — forma a espinha dorsal do pipeline de ingestão de dados, automatizando um processo que seria manual e demorado.

Seção 2.3: Tratamento Robusto de Erros e Exceções

O web scraping é uma atividade inerentemente instável. Sites mudam suas estruturas, implementam medidas anti-bot, e os dados alvo (como perfis de usuário) podem ser privados, restritos ou inexistentes. Uma aplicação de nível profissional não pode simplesmente assumir que todas as requisições serão bem-sucedidas. Ela deve ser arquitetada com a premissa de falhas parciais e ser capaz de tratá-las de forma elegante.

A biblioteca apify-client facilita isso ao levantar uma exceção específica, ApifyApiError, sempre que a API da Apify retorna um erro.[18] Ao envolver todas as chamadas de API em um bloco

try...except ApifyApiError, podemos construir uma aplicação resiliente.

Python

```
# Exemplo de tratamento de erro em src/data_ingestion/apify_handler.py
from apify_client.errors import ApifyApiError

def scrape_profile_data(client: ApifyClient, usernames: list[str]):
    """Executa o Instagram Profile Scraper com tratamento de erros."""
    print(f"Iniciando a extração de dados de perfil para: {usernames}")

    try:
        actor = client.actor("apify/instagram-profile-scraper")
        run_input = {"usernames": usernames}
```

```
        run = actor.call(run_input=run_input)
        print("Extração de dados de perfil concluída.")
        return run
    except ApifyApiError as e:
        print(f"Ocorreu um erro na API da Apify: {e.message}")
        print(f"  - Tipo do Erro: {e.type}")
        print(f"  - Status Code: {e.status_code}")
        # Retorna None para sinalizar a falha e permitir que o fluxo principal continue
        return None
```

Tratamento de Casos Específicos:
Um tratamento de erro genérico é bom, mas uma aplicação verdadeiramente robusta antecipa falhas comuns e reage de acordo.

- **Perfis Privados ou Inexistentes**: Este é um dos erros mais comuns. Quando um scraper tenta acessar um perfil que não é público ou que foi excluído, a API pode retornar um erro ou simplesmente não encontrar dados. A lógica da aplicação não deve travar. Em vez disso, ela deve capturar a falha, registrar uma mensagem informativa (ex: "Aviso: O perfil 'competidor_invalido' não pôde ser analisado. Pode ser privado ou não existir.") e continuar o processamento para os outros concorrentes da lista. A arquitetura deve ser projetada para tolerar a ausência de dados de um ou mais concorrentes.
- **Limites de Taxa (Rate Limits - HTTP 429)**: Uma grande vantagem do apify-client é que ele gerencia automaticamente as tentativas de repetição com um mecanismo de *exponential backoff* para erros de limite de taxa.[16] Isso significa que se a aplicação fizer muitas requisições em um curto período, o cliente esperará um tempo crescente antes de tentar novamente, aumentando significativamente a probabilidade de sucesso sem a necessidade de implementar essa lógica complexa manualmente. Explicar essa funcionalidade embutida destaca a robustez da ferramenta escolhida.
- **Erros de Validação de Schema (Schema Validation Errors)**: A API da Apify pode validar os dados que são enviados a ela. Se um Actor tentar enviar dados para um dataset que não correspondem a um schema pré-definido, a API retornará um erro 400 com detalhes sobre os itens inválidos.[20] Embora nosso projeto esteja principalmente
  puxando dados, compreender esse tipo de erro é crucial para um conhecimento completo do ecossistema da Apify e para construir Actors mais complexos no futuro.

A existência de documentação detalhada sobre ApifyApiError [18] e a funcionalidade de retentativa automática [16] não são apenas recursos convenientes; são um reconhecimento de que a extração de dados da web é um campo de batalha. Uma aplicação de nível especialista abraça essa realidade, programando defensivamente e construindo sistemas que não são apenas funcionais em condições ideais, mas resilientes diante da adversidade esperada.

Parte III: A Camada de Inteligência - Gerando Insights Estratégicos com LangChain

Esta é a seção onde os dados brutos são transformados em inteligência acionável. Utilizando o framework LangChain, orquestramos Large Language Models (LLMs) para ir além dos números, realizando análises qualitativas e estruturando a entrada e a saída de informações de forma inteligente e confiável.

Seção 3.1: Desconstruindo o Briefing com Processamento de Linguagem Natural

O ponto de partida para a análise é um "briefing" da empresa. Uma abordagem tradicional exigiria que o usuário inserisse os dados em campos de formulário rígidos ou como argumentos de linha de comando. Uma abordagem moderna e mais poderosa, no entanto, permite que o usuário forneça o briefing em linguagem natural, e a aplicação inteligentemente extrai as informações necessárias.

Para isso, combinamos o poder dos LLMs com a validação de dados do Pydantic e a orquestração do LangChain.

1. **Definição do Schema de Entrada com Pydantic**: Primeiro, definimos uma classe Pydantic que representa a estrutura de dados que queremos extrair do briefing. Esta classe serve como um "contrato" para o LLM.
   Python
   ```
   # Em src/analysis/engine.py
   from pydantic import BaseModel, Field
   from typing import List
   ```

```
class AnalysisBrief(BaseModel):
    """Estrutura de dados para o briefing da análise de concorrentes."""
    client_name: str = Field(description="O nome da empresa cliente para a qual o relatório está sendo gerado.")
    competitor_usernames: List[str] = Field(description="Uma lista de nomes de usuário do Instagram dos concorrentes a serem analisados.")
```

2. **Uso de with_structured_output do LangChain**: O método with_structured_output do LangChain é uma ferramenta poderosa que instrui um LLM a gerar uma saída que adere a um schema Pydantic específico.[22] Isso garante que a saída do modelo não seja apenas texto livre, mas um objeto JSON estruturado e validado, pronto para uso programático.

```
Python
# Em src/analysis/engine.py
from langchain_openai import ChatOpenAI

def parse_briefing(briefing_text: str) -> AnalysisBrief:
    """
    Usa um LLM para analisar um texto de briefing em linguagem natural e extrair
    informações estruturadas usando um schema Pydantic.
    """
    llm = ChatOpenAI(model="gpt-4o", temperature=0)
    structured_llm = llm.with_structured_output(AnalysisBrief)

    prompt = f"""
    Analise o seguinte texto de briefing e extraia as informações necessárias.
    O nome da empresa cliente é a empresa que solicita o relatório.
    Os concorrentes são os outros nomes de usuário mencionados.

    Briefing: "{briefing_text}"
    """

    try:
        parsed_brief = structured_llm.invoke(prompt)
        return parsed_brief
    except Exception as e:
        print(f"Falha ao analisar o briefing: {e}")
        return None
```

3. **Execução**: Agora, a aplicação pode aceitar uma entrada de texto simples do usuário e convertê-la em um objeto de dados utilizável.

```python
# Em main.py
user_briefing = """
Por favor, prepare um relatório de concorrentes para minha empresa, 'QuantumLeap Coffee'.
Quero analisar nossos principais concorrentes: 'AstroBeans', 'Chromatic Brews' e 'TheDailyGrind'.
"""
brief_data = parse_briefing(user_briefing)
if brief_data:
    print(f"Cliente: {brief_data.client_name}")
    print(f"Concorrentes: {brief_data.competitor_usernames}")
```

Esta abordagem não apenas melhora a experiência do usuário, tornando a interação mais natural, mas também demonstra uma aplicação avançada de LLMs para a extração de entidades e estruturação de dados a partir de texto não estruturado.[22]

Seção 3.2: Definição e Cálculo de Indicadores Chave de Desempenho (KPIs)

Para que a análise seja valiosa, ela deve ser fundamentada em métricas de marketing estabelecidas. Os Indicadores Chave de Desempenho (KPIs) fornecem uma base quantitativa para comparar a performance dos concorrentes. Utilizaremos a biblioteca pandas para carregar os dados JSON extraídos pela Apify em DataFrames, o que facilita enormemente a manipulação e o cálculo desses KPIs.

Os principais KPIs para análise de concorrentes no Instagram incluem:

- **Taxa de Crescimento de Seguidores**: Mede o ritmo com que um concorrente está atraindo novos seguidores. Embora uma única extração não permita calcular essa taxa, a estrutura de dados será projetada para armazenar o número de seguidores, permitindo o cálculo em execuções futuras e periódicas do relatório.[26]
- Taxa de Engajamento por Post: Uma das métricas mais importantes, mede a interação do público com um post específico. A fórmula mais comum é: $TaxadeEngajamento = \frac{Nu´merodeSeguidores}{(Curtidas + Comenta´rios)} \times 100.26$
- **Taxa de Engajamento Média**: A média das taxas de engajamento de todos os

posts analisados de um concorrente. Isso fornece um benchmark único e comparável. Para contextualizar, pode-se comparar esse valor com médias da indústria, como a mediana de 0.43% reportada pela RivalIQ.[27]

- Taxa de Engajamento de Reels: Os Reels frequentemente têm uma dinâmica de engajamento diferente e devem ser analisados separadamente. Uma fórmula mais abrangente pode ser usada, se os dados estiverem disponíveis: TaxadeEngajamentodeReels=Visualizac\c o~es(Curtidas+Comenta´rios+Compartilhamentos+Salvamentos)×100.28
- **Frequência de Postagem**: O número médio de posts por semana ou por mês, indicando o nível de atividade e consistência do concorrente.

A tabela a seguir serve como um guia de referência, conectando as métricas estratégicas aos dados brutos disponíveis.

Tabela 3: KPIs Essenciais para Análise de Concorrentes no Instagram

Nome do KPI	Fórmula / Método de Cálculo	Dados Necessários (da Apify)	Importância Estratégica
Taxa de Engajamento por Post	followersCount(likes Count+commentsCo unt)×100	likesCount, commentsCount (do Post Scraper), followersCount (do Profile Scraper)	Mede a ressonância de cada conteúdo individual com a audiência.
Taxa de Engajamento Média	Média da Taxa de Engajamento por Post	Calculada a partir dos resultados individuais.	Fornece um benchmark de performance geral para comparação entre concorrentes.
Frequência de Postagem	Contagem de posts em um período / número de dias no período	timestamp de cada post.	Indica a consistência e o volume da estratégia de conteúdo do concorrente.
Análise de Formato de Conteúdo	Contagem de posts por tipo (Image, Sidecar, Video)	type de cada post.	Revela o mix de formatos de conteúdo utilizado pelo concorrente.

A implementação desses cálculos em pandas permite uma análise quantitativa rápida e eficiente, transformando milhares de pontos de dados em um painel de controle claro sobre o desempenho dos concorrentes.

Seção 3.3: Análise de Conteúdo Avançada via Engenharia de Prompts

A análise quantitativa nos diz *o quão bem* um concorrente está performando, mas a análise qualitativa nos diz *o porquê*. Para extrair insights sobre a estratégia de conteúdo e comunicação, utilizaremos um LLM como um analista de marketing virtual. Isso nos permite identificar os pilares de conteúdo e o tom de voz dos concorrentes em escala.

1. Identificação de Pilares de Conteúdo
Pilares de conteúdo são os 3 a 5 temas ou tópicos centrais sobre os quais uma marca constrói sua comunicação de forma consistente.29 Em vez de um analista humano ler manualmente centenas de posts, podemos alimentar as legendas dos posts de maior engajamento de um concorrente a um LLM e pedir que ele identifique esses pilares.
2. Engenharia de Prompts Estratégica
A chave para obter resultados de alta qualidade de um LLM é a engenharia de prompts. Um prompt bem construído guia o modelo para a resposta desejada. Utilizaremos uma abordagem multifacetada 31:

- **Persona (Role)**: "Você é um estrategista de marketing de mídias sociais sênior, especializado em análise de conteúdo de marcas de consumo."
- **Contexto (Context)**: "Você está analisando o desempenho no Instagram de um concorrente chamado '[Nome do Concorrente]'. A seguir estão as legendas de seus posts mais recentes e com maior engajamento: [Inserir aqui uma concatenação das legendas dos posts]."
- **Tarefa (Task)**: "Com base nessas legendas, realize duas análises:
    1. **Pilares de Conteúdo**: Identifique os 3 a 5 principais pilares de conteúdo (temas ou categorias de assunto recorrentes). Agrupe ideias semelhantes em categorias amplas e dê a cada pilar um nome conciso e descritivo.
    2. **Tom de Voz**: Analise e descreva o tom de voz geral da comunicação. Seja específico (ex: 'Inspirador e motivacional com um toque de humor', 'Educacional e profissional, mas acessível', 'Irreverente e jovem')."
- **Especificação de Formato (Format Specification)**: "Sua resposta deve ser exclusivamente um objeto JSON que se conforme ao seguinte schema Pydantic.

Não inclua nenhum texto ou explicação fora do objeto JSON.".[34]

3. Garantia de Saída Estruturada
Para garantir que a resposta do LLM seja sempre um JSON limpo e utilizável, definimos um schema Pydantic e usamos novamente o with_structured_output do LangChain.23

Python

```
# Em src/analysis/engine.py
from typing import List

class ContentStrategyAnalysis(BaseModel):
    """Análise da estratégia de conteúdo e tom de voz de um concorrente."""
    content_pillars: List[str] = Field(description="Uma lista de 3 a 5 pilares de conteúdo identificados.")
    tone_of_voice: str = Field(description="Uma descrição detalhada do tom de voz da marca.")
    summary: str = Field(description="Um resumo executivo da estratégia de conteúdo geral.")

def analyze_content_strategy(captions: List[str], competitor_name: str) -> ContentStrategyAnalysis:
    """Usa um LLM para analisar legendas e identificar pilares de conteúdo e tom de voz."""
    llm = ChatOpenAI(model="gpt-4o", temperature=0)
    structured_llm = llm.with_structured_output(ContentStrategyAnalysis)

    # Construção do prompt detalhado...
    prompt = f"""
    Persona: Você é um estrategista de marketing de mídias sociais sênior...
    Contexto: Você está analisando o concorrente '{competitor_name}'. Legendas: {captions}
    Tarefa: Analise os Pilares de Conteúdo e o Tom de Voz.
    Formato: Responda apenas com o objeto JSON.
    """

    analysis = structured_llm.invoke(prompt)
    return analysis
```

A combinação da análise quantitativa de KPIs com esta análise qualitativa impulsionada por LLM cria uma visão holística e poderosa. Um KPI pode mostrar que a taxa de engajamento de um concorrente é de 2.1%, mas é a análise de conteúdo que revela que esse engajamento é impulsionado principalmente por seu pilar de

"dicas de sustentabilidade", enquanto seu pilar de "promoções de produtos" tem um desempenho muito inferior. Este é o tipo de insight diretamente acionável que pode e deve informar a estratégia de conteúdo e comunicação de uma empresa.

Seção 3.4: Construindo um Agente LangChain para Análise (Extensão Opcional)

A abordagem descrita até agora segue um fluxo linear: extrair dados, analisar KPIs, analisar conteúdo, gerar relatório. Para uma flexibilidade ainda maior, o projeto poderia ser reestruturado para usar um "Agente" LangChain. Um agente é um sistema que utiliza um LLM para decidir qual sequência de ações tomar, escolhendo a partir de um conjunto de ferramentas disponíveis.[39]

Implementação Conceitual:

1. **Definição de Ferramentas (Tools)**: Cada função principal do nosso pipeline seria encapsulada como uma ferramenta usando o decorador @tool do LangChain.[39]
   Python
   ```
   from langchain_core.tools import tool

   @tool
   def get_competitor_profile_data(usernames: List[str]) -> dict:
       """Busca dados de perfil de alto nível para os nomes de usuário do Instagram fornecidos."""
       # Lógica para chamar o apify/instagram-profile-scraper
       ...
       return profile_data

   @tool
   def analyze_competitor_engagement(username: str) -> dict:
       """Calcula KPIs de engajamento para um único concorrente."""
       # Lógica para chamar o post scraper e calcular KPIs
       ...
       return engagement_kpis
   ```
2. **Criação do Agente**: Um agente seria construído usando um construtor pré-definido como create_react_agent, que recebe o LLM e a lista de ferramentas.[39]
3. **Execução Flexível**: Com o agente implementado, o usuário poderia fazer

perguntas mais complexas e em linguagem natural, e o agente determinaria as ferramentas a serem chamadas e a ordem de execução.

- **Consulta do Usuário**: "Compare a taxa de engajamento média da 'AstroBeans' com a da 'Chromatic Brews' e depois me dê um resumo dos pilares de conteúdo da 'AstroBeans'."
- **Plano do Agente (pensamento do LLM)**:
  1. Preciso da taxa de engajamento de 'AstroBeans'. Vou usar a ferramenta analyze_competitor_engagement com o input username='AstroBeans'.
  2. Preciso da taxa de engajamento de 'Chromatic Brews'. Vou usar a ferramenta analyze_competitor_engagement com o input username='Chromatic Brews'.
  3. Preciso dos pilares de conteúdo de 'AstroBeans'. Vou usar a ferramenta analyze_content_strategy (uma outra ferramenta a ser criada) com os posts da 'AstroBeans'.
  4. Com todos os resultados, vou sintetizar a resposta final para o usuário.

Embora a implementação completa de um agente esteja fora do escopo do projeto principal para manter a clareza, delinear essa arquitetura demonstra o caminho para uma evolução futura e uma compreensão de conceitos mais avançados do LangChain.

Parte IV: A Camada de Apresentação - Criando Relatórios Profissionais com python-docx

Esta parte final do pipeline concentra-se na tradução dos dados e insights analisados em um artefato tangível e profissional: um documento Word (.docx) com design polido e personalizado. O desafio aqui é superar as limitações da biblioteca python-docx para entregar um resultado que atenda à exigência de "design profissional e personalizado".

Seção 4.1: Dominando a Estilização de Documentos com Templates

A criação de estilos complexos de parágrafos e tabelas de forma programática usando python-docx é, na melhor das hipóteses, impraticável e, na pior, impossível para a maioria dos casos de uso complexos. A API de alto nível não expõe métodos para definir, por exemplo, bordas de tabela personalizadas, preenchimentos de células complexos ou fontes específicas de forma granular e reutilizável como um "estilo".[7]

A solução para essa limitação fundamental não está no código, mas na arquitetura do fluxo de trabalho. A prática recomendada e mais poderosa é a **abordagem "template-first"**.[7]

1. **Criação do Template (template.docx)**: O trabalho de design é feito na ferramenta mais adequada para isso: o Microsoft Word. Um documento, que chamaremos de template.docx, é criado e salvo no diretório templates/. Dentro deste arquivo, todos os estilos necessários para o relatório são pré-definidos:
   - **Estilos de Parágrafo**: Estilos personalizados para o título do relatório, cabeçalhos de seção (Heading 1, Heading 2), corpo de texto (BodyTextCustom), citações (QuoteCustom), etc.
   - **Estilos de Tabela**: O mais importante é a criação de um estilo de tabela personalizado (ex: CustomCompetitorTableStyle). Neste estilo, definimos as fontes, tamanhos de fonte, cores de texto, cores de fundo para a linha de cabeçalho e linhas de dados, espessura e cor das bordas, e alinhamento do texto nas células.[7]
2. **Carregando o Template no Python**: Em vez de criar um documento em branco com document = Document(), a aplicação inicia o processo de geração do relatório carregando este template. Isso torna todos os estilos personalizados definidos no arquivo .docx disponíveis para uso programático por meio de seus nomes.[7]

```Python
# Em src/reporting/generator.py
from docx import Document

def create_report_from_template(template_path: str):
    """
    Inicia a criação de um documento Word carregando um template com estilos pré-definidos.
    """
    try:
        document = Document(template_path)
        return document
```

```
    except Exception as e:
        print(f"Erro ao carregar o template '{template_path}': {e}")
        # Como fallback, cria um documento em branco
        return Document()
```

Esta abordagem de separar a definição de estilo (feita no Word) da geração de conteúdo (feita no Python) é a chave para produzir documentos com aparência profissional. Ela aproveita o melhor de ambos os mundos: o poder de design visual do Word e a automação de conteúdo do Python.

Seção 4.2: Geração Programática de Elementos do Relatório

Com o documento carregado a partir do template, a adição de conteúdo utiliza as funções padrão da biblioteca python-docx, mas com a capacidade de aplicar os estilos personalizados.

- **Títulos e Parágrafos**: document.add_heading() e document.add_paragraph() são usados para adicionar texto. O parâmetro style é usado para aplicar os estilos pré-definidos do nosso template.[42]
  ```
  Python
  # Em src/reporting/generator.py
  document.add_heading('Relatório de Análise de Concorrentes', level=0) # Usa o estilo 'Title'
  document.add_heading('1. Visão Geral dos Concorrentes', level=1) # Usa o estilo 'Heading 1'
  document.add_paragraph(
      "Esta seção apresenta uma análise comparativa de alto nível...",
      style='BodyTextCustom'
  )
  document.add_page_break() # Adiciona uma quebra de página
  ```

- **Tabelas**: As tabelas são criadas dinamicamente com document.add_table() e preenchidas com os dados dos nossos DataFrames pandas.[45] O estilo de tabela personalizado é aplicado no momento da criação ou posteriormente.
  ```
  Python
  # Em src/reporting/generator.py
  ```

```
# kpi_df é um DataFrame pandas com os dados dos concorrentes
table = document.add_table(rows=1, cols=len(kpi_df.columns))
table.style = 'CustomCompetitorTableStyle' # Aplica nosso estilo personalizado

# Adiciona o cabeçalho da tabela
hdr_cells = table.rows.cells
for i, col_name in enumerate(kpi_df.columns):
    hdr_cells[i].text = col_name

# Adiciona as linhas de dados
for index, row in kpi_df.iterrows():
    row_cells = table.add_row().cells
    for i, cell_value in enumerate(row):
        row_cells[i].text = str(cell_value)
```

Este fluxo de trabalho básico permite a montagem estruturada do relatório, aplicando consistentemente a identidade visual definida no template.

Seção 4.3: Formatação Avançada de Tabelas e Células

Para alcançar um nível ainda maior de personalização, como destacar células específicas com cores (por exemplo, colorir a célula com a maior taxa de engajamento), é necessário ir além da API de alto nível do python-docx e manipular diretamente o XML subjacente (OOXML).

O Desafio: A API não oferece um método simples como cell.background_color = '#FF0000'.[48]

A Solução: Manipulação de OOXML: Para alterar a cor de fundo de uma célula, criamos uma função auxiliar que interage com os elementos XML. Esta técnica, embora mais complexa, oferece controle total sobre a formatação.[49]

Python

```python
# Em src/reporting/generator.py (função auxiliar)
from docx.oxml.ns import qn
from docx.oxml import OxmlElement

def set_cell_shading(cell, hex_color: str):
    """
    Define a cor de fundo de uma célula da tabela.
    hex_color: Uma string com o código de cor hexadecimal RGB (ex: 'A9A9A9').
    """
    try:
        shading_elm = OxmlElement('w:shd')
        shading_elm.set(qn('w:fill'), hex_color)
        cell._tc.get_or_add_tcPr().append(shading_elm)
    except Exception as e:
        print(f"Não foi possível aplicar a cor de fundo à célula: {e}")
```

Estilizando o Texto do Cabeçalho: Da mesma forma, para aplicar negrito ao texto em células específicas (como no cabeçalho da tabela), não se pode simplesmente definir cell.text.bold = True. É preciso acessar o parágrafo dentro da célula e adicionar uma "run" de texto com a propriedade de negrito ativada.[52]

Python

```python
# Em src/reporting/generator.py
def set_header_text_bold(table):
    """Aplica negrito ao texto na primeira linha (cabeçalho) de uma tabela."""
    for cell in table.rows.cells:
        # Limpa o texto existente, se houver
        text = cell.text
        cell.text = ""
        # Adiciona o texto de volta em uma run com negrito
        p = cell.paragraphs
        run = p.add_run(text)
        run.bold = True
        p.alignment = 1 # Centraliza o parágrafo
```

Essas funções auxiliares encapsulam a complexidade da manipulação de XML, permitindo que o resto do código de geração do relatório as utilize de forma limpa para aplicar formatações dinâmicas e personalizadas.

Seção 4.4: Integrando Visualizações de Dados

Um relatório de dados é imensamente enriquecido pela inclusão de gráficos e visualizações, que podem transmitir padrões e comparações de forma muito mais eficaz do que tabelas de números. Utilizaremos as bibliotecas matplotlib e seaborn para criar os gráficos e uma técnica eficiente para inseri-los no documento Word.

1. Criação das Visualizações: Com os dados processados em DataFrames pandas, podemos criar uma variedade de gráficos relevantes [54]:

- Um gráfico de barras comparando as taxas de engajamento médias dos concorrentes.
- Um gráfico de pizza mostrando a distribuição dos formatos de conteúdo (Imagem, Carrossel, Vídeo) para um concorrente específico.
- Um gráfico de linhas mostrando o crescimento de seguidores (se dados históricos estiverem disponíveis).

2. Salvando Gráficos em Memória: A abordagem padrão de salvar o gráfico como um arquivo de imagem no disco (plt.savefig('chart.png')) e depois lê-lo para inserir no documento é ineficiente e desordenada. Uma técnica muito superior é salvar o gráfico diretamente em um fluxo de bytes na memória usando o módulo io.[56]

Python

```
# Em src/reporting/generator.py
import matplotlib.pyplot as plt
import seaborn as sns
import io
from docx.shared import Inches
```

```
def create_engagement_chart(df, output_buffer):
    """Cria um gráfico de barras de engajamento e o salva em um buffer de memória."""
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Competitor', y='Avg Engagement Rate', data=df)
    plt.title('Comparativo de Taxa de Engajamento Média')
    plt.ylabel('Taxa de Engajamento Média (%)')
    plt.xlabel('Concorrente')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()

    # Salva o gráfico no buffer em vez de um arquivo
    plt.savefig(output_buffer, format='png')
    plt.close() # Fecha a figura para liberar memória

    # Reposiciona o ponteiro do buffer para o início
    output_buffer.seek(0)
```

3. Inserindo a Imagem no Documento: O objeto de buffer de memória pode então ser passado diretamente para o método document.add_picture(). Isso evita a criação de arquivos temporários e mantém o processo de geração do relatório autocontido e limpo.[42]

Python

```
# Em src/reporting/generator.py
#... dentro da função principal de geração de relatório...

image_buffer = io.BytesIO()
create_engagement_chart(kpi_summary_df, image_buffer)

document.add_picture(image_buffer, width=Inches(6.0))
```

A seguir, exemplos das tabelas que serão geradas dinamicamente no relatório final, servindo como a espinha dorsal da apresentação dos dados.

Tabela 4 (Exemplo de Saída Gerada): Painel de Desempenho dos Concorrentes

Concorrente	Seguidores	Seguindo	Total de Posts	Média de Curtidas/Post	Média de Comentários/Post	Taxa de Engajamento Média (%)
AstroBeans	150,432	350	1,204	3,012	150	2.10
Chromatic Brews	89,765	501	876	987	88	1.20
TheDaily Grind	250,110	1,024	2,510	2,500	125	1.05
Quantum Leap Coffee	50,234	430	543	1,255	110	2.72

Nota: A linha de cabeçalho seria estilizada com cor de fundo e negrito programaticamente.

Tabela 5 (Exemplo de Saída Gerada): Análise de Pilares de Conteúdo para 'AstroBeans'

Pilar de Conteúdo	Formato Principal Utilizado	Taxa de Engajamento Média para este Pilar (%)
Educação sobre Café	Carrossel (Imagens)	2.55
Bastidores da Torrefação	Reels (Vídeo)	2.90
Depoimentos de Clientes	Imagem Única	1.80
Promoções e Lançamentos	Imagem Única	1.15

Essas tabelas, combinadas com as visualizações de dados, transformam os dados brutos e as análises de LLM em uma narrativa coesa e fácil de digerir, fornecendo insights estratégicos de forma clara e direta no relatório final.

Parte V: Código Completo do Projeto e Guia de Implantação

Esta parte final consolida todo o trabalho, apresentando o código-fonte completo e comentado da aplicação, juntamente com um guia claro para configuração e execução.

Seção 5.1: O Orquestrador Principal (main.py)

Este é o script de entrada que amarra todos os módulos. Ele é responsável por carregar o ambiente, obter o briefing do usuário, invocar os módulos de ingestão, análise e, finalmente, de geração de relatórios.

Python

```
# main.py
import os
import json
from datetime import datetime
from dotenv import load_dotenv

from src.data_ingestion import apify_handler
from src.analysis import engine
from src.reporting import generator
from config import settings

def main():
    """
    Função principal que orquestra o fluxo completo de geração do relatório
    de análise de concorrentes.
    """
    print("Iniciando o processo de geração de relatório...")
    load_dotenv()
```

```
    # 1. Obter o briefing do usuário (aqui hardcoded, mas poderia ser um input)
    user_briefing = """
    Por favor, prepare um relatório de análise de concorrentes para a minha empresa, 'QuantumLeap Coffee'.
    Quero uma análise detalhada dos meus principais concorrentes no Instagram: 'AstroBeans', 'Chromatic Brews' e 'TheDailyGrind'.
    """

    # 2. Analisar o briefing com LangChain para extrair informações estruturadas
    print("Analisando o briefing...")
    brief_data = engine.parse_briefing(user_briefing)
    if not brief_data:
        print("Não foi possível analisar o briefing. Encerrando.")
        return

    client_name = brief_data.client_name
    competitor_usernames = brief_data.competitor_usernames
    all_profiles_to_scan = competitor_usernames + [client_name.lower().replace(" ", "")]

    print(f"Cliente: {client_name}")
    print(f"Concorrentes a serem analisados: {competitor_usernames}")

    # 3. Ingestão de Dados via Apify
    print("\nIniciando a ingestão de dados da Apify...")
    apify_client = apify_handler.get_apify_client()

    # Extrair dados de perfil
    profile_run = apify_handler.scrape_profile_data(apify_client, all_profiles_to_scan)
    profile_data_path = os.path.join(settings.RAW_DATA_PATH, "profile_data.json")
    profile_data = apify_handler.get_data_from_run(apify_client, profile_run, profile_data_path)

    # Extrair dados de posts
    post_run = apify_handler.scrape_post_data(apify_client, all_profiles_to_scan, max_posts=settings.MAX_POSTS_PER_PROFILE)
    post_data_path = os.path.join(settings.RAW_DATA_PATH, "post_data.json")
    post_data = apify_handler.get_data_from_run(apify_client, post_run, post_data_path)
```

```
    if not profile_data or not post_data:
        print("Falha na coleta de dados da Apify. Encerrando.")
        return

    # 4. Análise dos Dados
    print("\nIniciando a análise dos dados...")
    # Carregar dados em dataframes
    profile_df = engine.load_profiles_to_df(profile_data_path)
    posts_df = engine.load_posts_to_df(post_data_path)

    # Calcular KPIs
    kpi_df = engine.calculate_kpis(profile_df, posts_df)
    print("KPIs calculados:")
    print(kpi_df)

    # Análise de conteúdo com LLM
    content_analysis_results = {}
    for username in all_profiles_to_scan:
        print(f"Analisando estratégia de conteúdo para: {username}")
        analysis = engine.analyze_content_strategy_for_user(posts_df, username)
        content_analysis_results[username] = analysis

    # 5. Geração do Relatório
    print("\nIniciando a geração do relatório.docx...")
    report_filename = f"Relatorio_Concorrentes_{client_name.replace(' ',
'_')}_{datetime.now().strftime('%Y%m%d')}.docx"
    report_path = os.path.join(settings.REPORTS_PATH, report_filename)

    generator.generate_full_report(
        client_name=client_name,
        kpi_df=kpi_df,
        content_analysis=content_analysis_results,
        profile_df=profile_df,
        posts_df=posts_df,
        output_path=report_path,
        template_path=settings.TEMPLATE_PATH
    )
```

```
    print(f"\nRelatório gerado com sucesso em: {report_path}")

if __name__ == "__main__":
    # Garante que os diretórios de dados e relatórios existam
    os.makedirs(settings.RAW_DATA_PATH, exist_ok=True)
    os.makedirs(settings.REPORTS_PATH, exist_ok=True)
    main()
```

Seção 5.2: Módulos de Utilidade (Código-Fonte Completo)

A seguir, o código completo e comentado para os módulos que compõem o núcleo da aplicação.

src/data_ingestion/apify_handler.py

Python

```
# src/data_ingestion/apify_handler.py
import os
import json
from apify_client import ApifyClient
from apify_client.errors import ApifyApiError

def get_apify_client() -> ApifyClient:
    """Inicializa e retorna uma instância do cliente da Apify."""
    apify_token = os.getenv("APIFY_API_TOKEN")
    if not apify_token:
        raise ValueError("Token da API da Apify não encontrado. Verifique seu arquivo.env.")
    return ApifyClient(apify_token)

def scrape_profile_data(client: ApifyClient, usernames: list[str]) -> dict:
    """Executa o Instagram Profile Scraper para uma lista de nomes de usuário com tratamento de erros."""
```

```
    print(f"Iniciando a extração de dados de perfil para: {usernames}")
    try:
        actor = client.actor("apify/instagram-profile-scraper")
        run_input = {"usernames": usernames}
        run = actor.call(run_input=run_input)
        print("Extração de dados de perfil concluída.")
        return run
    except ApifyApiError as e:
        print(f"Ocorreu um erro na API da Apify ao buscar perfis: {e.message}")
        return None

def scrape_post_data(client: ApifyClient, usernames: list[str], max_posts: int) -> dict:
    """Executa o Instagram Post Scraper com tratamento de erros."""
    print(f"Iniciando a extração de até {max_posts} posts para: {usernames}")
    try:
        actor = client.actor("apify/instagram-post-scraper")
        run_input = {"usernames": usernames, "resultsLimit": max_posts}
        run = actor.call(run_input=run_input)
        print("Extração de posts concluída.")
        return run
    except ApifyApiError as e:
        print(f"Ocorreu um erro na API da Apify ao buscar posts: {e.message}")
        return None

def get_data_from_run(client: ApifyClient, run: dict, output_path: str) -> list:
    """Recupera os itens de um dataset de uma execução e salva em um arquivo JSON."""
    if not run:
        print("Execução inválida, não é possível buscar dados.")
        return

    dataset_id = run.get("defaultDatasetId")
    if not dataset_id:
        print("Nenhum dataset encontrado na execução.")
        return

    print(f"Recuperando dados do dataset: {dataset_id}")
    try:
        dataset_items = client.dataset(dataset_id).list_items().items
        os.makedirs(os.path.dirname(output_path), exist_ok=True)
```

```
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(dataset_items, f, ensure_ascii=False, indent=4)
        print(f"Dados salvos em: {output_path}")
        return dataset_items
    except ApifyApiError as e:
        print(f"Erro ao buscar itens do dataset {dataset_id}: {e.message}")
        return
```

src/analysis/engine.py

Python

```
# src/analysis/engine.py
import pandas as pd
from typing import List
from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI

# --- Pydantic Schemas ---
class AnalysisBrief(BaseModel):
    client_name: str = Field(description="O nome da empresa cliente.")
    competitor_usernames: List[str] = Field(description="Lista de nomes de usuário dos concorrentes.")

class ContentStrategyAnalysis(BaseModel):
    content_pillars: List[str] = Field(description="Lista de 3-5 pilares de conteúdo.")
    tone_of_voice: str = Field(description="Descrição do tom de voz.")
    summary: str = Field(description="Resumo da estratégia de conteúdo.")

# --- Funções de Análise ---
def parse_briefing(briefing_text: str) -> AnalysisBrief:
    llm = ChatOpenAI(model="gpt-4o", temperature=0)
    structured_llm = llm.with_structured_output(AnalysisBrief)
    prompt = f"Analise o briefing a seguir e extraia as informações necessárias. Briefing: \"{briefing_text}\""
    try:
```

```
        return structured_llm.invoke(prompt)
    except Exception as e:
        print(f"Falha ao analisar o briefing: {e}")
        return None

def load_profiles_to_df(path: str) -> pd.DataFrame:
    return pd.read_json(path)

def load_posts_to_df(path: str) -> pd.DataFrame:
    df = pd.read_json(path)
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    return df

def calculate_kpis(profile_df: pd.DataFrame, posts_df: pd.DataFrame) -> pd.DataFrame:
    # Agrega dados de posts por usuário
    post_metrics = posts_df.groupby('ownerUsername').agg(
        avg_likes=('likesCount', 'mean'),
        avg_comments=('commentsCount', 'mean'),
        total_posts=('id', 'count')
    ).reset_index()

    # Junta com dados de perfil
    merged_df = pd.merge(profile_df, post_metrics, left_on='username',
right_on='ownerUsername', how='left')

    # Calcula taxa de engajamento
    merged_df['avg_engagement_rate'] = ((merged_df['avg_likes'] +
merged_df['avg_comments']) / merged_df['followersCount']) * 100

    # Seleciona e renomeia colunas
    kpi_df = merged_df[[
        'username', 'followersCount', 'followsCount', 'postsCount',
        'avg_likes', 'avg_comments', 'avg_engagement_rate'
    ]].copy()
    kpi_df.columns =
    kpi_df = kpi_df.round(2)
    return kpi_df

def analyze_content_strategy_for_user(posts_df: pd.DataFrame, username: str) ->
```

```
ContentStrategyAnalysis:
    user_posts = posts_df[posts_df['ownerUsername'] == username]
    # Pega as legendas dos 10 posts com mais curtidas
    top_captions = user_posts.nlargest(10, 'likesCount')['caption'].dropna().tolist()

    if not top_captions:
        return ContentStrategyAnalysis(content_pillars=, tone_of_voice="N/A",
summary="Dados insuficientes para análise.")

    captions_text = "\n".join(f"- {c}" for c in top_captions)

    llm = ChatOpenAI(model="gpt-4o", temperature=0)
    structured_llm = llm.with_structured_output(ContentStrategyAnalysis)

    prompt = f"""
    Persona: Você é um estrategista de marketing de mídias sociais sênior.
    Contexto: Analise as seguintes legendas do perfil do Instagram '{username}':
    {captions_text}
    Tarefa: Identifique os 3-5 principais pilares de conteúdo e descreva o tom de voz da marca.
    Formato: Responda apenas com o objeto JSON.
    """
    try:
        return structured_llm.invoke(prompt)
    except Exception as e:
        print(f"Falha na análise de conteúdo para {username}: {e}")
        return ContentStrategyAnalysis(content_pillars=, tone_of_voice="Erro na análise",
summary="Erro na análise.")
```

src/reporting/generator.py

```
    Python


# src/reporting/generator.py
import io
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
from docx import Document
from docx.shared import Inches, Pt, RGBColor
from docx.oxml.ns import qn
from docx.oxml import OxmlElement

# --- Funções Auxiliares de Estilização ---
def set_cell_shading(cell, hex_color: str):
    """Define a cor de fundo de uma célula da tabela."""
    try:
        shading_elm = OxmlElement('w:shd')
        shading_elm.set(qn('w:fill'), hex_color)
        cell._tc.get_or_add_tcPr().append(shading_elm)
    except Exception as e:
        print(f"Não foi possível aplicar a cor de fundo à célula: {e}")

def set_header_style(table):
    """Aplica cor de fundo cinza e negrito ao cabeçalho da tabela."""
    for cell in table.rows.cells:
        set_cell_shading(cell, 'D9D9D9') # Cinza claro
        text = cell.text
        cell.text = ""
        p = cell.paragraphs
        run = p.add_run(text)
        run.bold = True
        run.font.color.rgb = RGBColor(0, 0, 0)
        p.alignment = 1 # WD_ALIGN_PARAGRAPH.CENTER

# --- Funções de Geração de Gráficos ---
def create_engagement_chart(df: pd.DataFrame) -> io.BytesIO:
    """Cria um gráfico de barras de engajamento e o retorna como um buffer de memória."""
    buffer = io.BytesIO()
    plt.figure(figsize=(10, 6))
    sns.set_style("whitegrid")
    bar_plot = sns.barplot(x='Perfil', y='Taxa de Engajamento Média (%)', data=df,
palette='viridis')
    plt.title('Comparativo de Taxa de Engajamento Média', fontsize=16)
    plt.ylabel('Taxa de Engajamento (%)', fontsize=12)
    plt.xlabel('Perfil', fontsize=12)
```

```
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(buffer, format='png', dpi=300)
    plt.close()
    buffer.seek(0)
    return buffer

# --- Função Principal de Geração de Relatório ---
def generate_full_report(client_name, kpi_df, content_analysis, profile_df, posts_df, output_path,
template_path):
    """Gera o relatório completo em.docx."""
    document = Document(template_path)

    # Título
    document.add_heading(f"Análise de Concorrentes do Instagram para {client_name}",
level=0)

    # Seção 1: Painel de KPIs
    document.add_heading("Painel Comparativo de KPIs", level=1)
    document.add_paragraph("A tabela a seguir resume os principais indicadores de desempenho
para cada perfil analisado.")

    # Adiciona tabela de KPIs
    table = document.add_table(rows=1, cols=len(kpi_df.columns))
    table.style = 'Table Grid' # Usar um estilo base, a formatação será aplicada por cima

    # Cabeçalho
    for i, col_name in enumerate(kpi_df.columns):
        table.cell(0, i).text = col_name

    # Dados
    for index, row in kpi_df.iterrows():
        row_cells = table.add_row().cells
        for i, cell_value in enumerate(row):
            row_cells[i].text = str(cell_value)

    set_header_style(table)

    # Adiciona gráfico de engajamento
```

```
    document.add_paragraph("\nVisualização da Taxa de Engajamento:", style='Intense Quote')
    chart_buffer = create_engagement_chart(kpi_df)
    document.add_picture(chart_buffer, width=Inches(6.5))

    document.add_page_break()

    # Seção 2: Análise Individual de Concorrentes
    document.add_heading("Análise Detalhada por Perfil", level=1)

    for username, analysis in content_analysis.items():
        document.add_heading(f"Análise de: {username}", level=2)

        # Resumo da Estratégia
        document.add_paragraph("Resumo da Estratégia:",
style='BodyTextCustom').add_run().bold = True
        document.add_paragraph(analysis.summary, style='List Bullet')

        # Tom de Voz
        document.add_paragraph("Tom de Voz:", style='BodyTextCustom').add_run().bold =
True
        document.add_paragraph(analysis.tone_of_voice, style='List Bullet')

        # Pilares de Conteúdo
        document.add_paragraph("Principais Pilares de Conteúdo:",
style='BodyTextCustom').add_run().bold = True
        for pillar in analysis.content_pillars:
            document.add_paragraph(pillar, style='List Bullet 2')

        document.add_paragraph("\n")

    # Salva o documento
    document.save(output_path)
```

Seção 5.3: O README.md do GitHub

Um README.md de alta qualidade é essencial para qualquer projeto de portfólio. Ele

deve ser claro, conciso e fornecer todas as informações necessárias para que outra pessoa possa entender, configurar e executar o projeto.

Gerador de Relatório de Análise de Concorrentes do Instagram com IA

Este projeto é uma ferramenta de automação que gera um relatório de análise de concorrentes para o Instagram. Utilizando a plataforma Apify para extração de dados, LangChain e modelos de linguagem (LLMs) para análise, e python-docx para a criação de relatórios, esta aplicação transforma um simples briefing em um documento .docx profissional e personalizado.

✨ Funcionalidades

- **Análise de Briefing em Linguagem Natural**: Extrai o nome do cliente e a lista de concorrentes de um texto simples.
- **Extração de Dados Abrangente**: Coleta dados de perfil (seguidores, posts, etc.) e dados de posts individuais (curtidas, comentários, legendas) usando a Apify.
- **Análise Quantitativa (KPIs)**: Calcula automaticamente os principais indicadores de desempenho, como taxa de engajamento média.
- **Análise Qualitativa com IA**: Utiliza um LLM para identificar os pilares de conteúdo e o tom de voz dos concorrentes.
- **Geração de Relatório Profissional**: Cria um documento .docx com design personalizado, incluindo texto, tabelas formatadas e visualizações de dados.
- **Segurança e Boas Práticas**: Gerencia chaves de API de forma segura e segue uma estrutura de projeto organizada.

🚀 Configuração e Instalação

Siga estes passos para configurar e executar o projeto em sua máquina local.

1. Clone o Repositório

Bash

```
git clone https://github.com/seu-usuario/seu-repositorio.git
cd seu-repositorio
```

2. Crie e Ative um Ambiente Virtual

É altamente recomendável usar um ambiente virtual para isolar as dependências do projeto.

Bash

```
python -m venv venv
# No Windows:
venv\Scripts\activate
# No macOS/Linux:
source venv/bin/activate
```

3. Instale as Dependências

Instale todas as bibliotecas necessárias a partir do arquivo requirements.txt.

Bash

```
pip install -r requirements.txt
```

4. Configure suas Chaves de API

Este projeto requer chaves de API para a Apify e para a OpenAI (ou outro provedor de LLM).

- Crie um arquivo chamado .env na raiz do projeto.
- Adicione suas chaves ao arquivo no seguinte formato:

```
#.env
APIFY_API_TOKEN="seu_token_da_apify"
OPENAI_API_KEY="seu_token_da_openai"
```

Importante: O arquivo .env já está listado no .gitignore para garantir que suas chaves secretas não sejam enviadas para o GitHub.

5. Adicione um Template de Documento

Para um design personalizado, crie um documento Word com seus estilos de parágrafo e tabela preferidos. Salve-o como template.docx dentro do diretório templates/.

⚙️ Uso

Para gerar um relatório, basta executar o script main.py.

```
Bash
```

```
python main.py
```

O script irá:

1. Analisar o briefing definido em main.py.

2. Executar os scrapers da Apify para coletar os dados.
3. Processar e analisar os dados.
4. Gerar um relatório .docx e salvá-lo no diretório reports/.

Você pode personalizar o briefing diretamente no arquivo main.py para analisar diferentes empresas e concorrentes.

📄 Exemplo de Saída

Após a execução, um arquivo como Relatorio_Concorrentes_QuantumLeap_Coffee_20240915.docx será gerado, contendo uma análise detalhada com tabelas e gráficos.

(Aqui você pode adicionar uma captura de tela do relatório gerado)

Referências citadas

1. How to organize your Python data science project · GitHub, acessado em junho 22, 2025, https://gist.github.com/ericmjl/27e50331f24db3e8f957d1fe7bbbe510
2. csymvoul/python-structure-template: A Python project to use as a template when developing a Python application - GitHub, acessado em junho 22, 2025, https://github.com/csymvoul/python-structure-template
3. Manage dependencies using requirements.txt | Aqua Documentation - JetBrains, acessado em junho 22, 2025, https://www.jetbrains.com/help/aqua/managing-dependencies.html
4. Best Practices for Managing Python Dependencies - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/python/best-practices-for-managing-python-dependencies/
5. Using Python Environment Variables with Python Dotenv - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/python/using-python-environment-variables-with-python-dotenv/
6. API Keys - Using Environment Variables in Python Projects - Free Video Tutorial, acessado em junho 22, 2025, https://www.nobledesktop.com/learn/python/api-keys-using-environment-variables-in-python-projects
7. How do i set table style · Issue #9 · python-openxml/python-docx - GitHub, acessado em junho 22, 2025, https://github.com/python-openxml/python-docx/issues/9

8. python-dotenv·PyPI, acessado em junho 22, 2025, https://pypi.org/project/python-dotenv/
9. Using Py Dotenv (python-dotenv) Package to Manage Env Variables - Configu, acessado em junho 22, 2025, https://configu.com/blog/using-py-dotenv-python-dotenv-package-to-manage-env-variables/
10. How to scrape data from Instagram: profiles, comments, posts and photos - Apify Blog, acessado em junho 22, 2025, https://blog.apify.com/scrape-instagram-posts-comments-and-more-21d05506aeb3/
11. Instagram Profile Scraper · Apify, acessado em junho 22, 2025, https://apify.com/apify/instagram-profile-scraper
12. Instagram Post Scraper · Apify, acessado em junho 22, 2025, https://apify.com/apify/instagram-post-scraper
13. Instagram Hashtag Scraper - Apify, acessado em junho 22, 2025, https://apify.com/apify/instagram-hashtag-scraper
14. Scrape Instagram hashtags - Apify, acessado em junho 22, 2025, https://apify.com/store/hashtag-scraper/scrape-instagram-hashtags
15. Instagram Scraper (Pay Per Result) - Apify, acessado em junho 22, 2025, https://apify.com/apidojo/instagram-scraper
16. apify-client - PyPI, acessado em junho 22, 2025, https://pypi.org/project/apify-client/
17. Dataset | Platform - Apify Documentation, acessado em junho 22, 2025, https://docs.apify.com/platform/storage/dataset
18. Error handling | API client for Python - Apify Documentation, acessado em junho 22, 2025, https://docs.apify.com/api/client/python/docs/concepts/error-handling
19. ApifyApiError | API | API client for Python - Apify Documentation, acessado em junho 22, 2025, https://docs.apify.com/api/client/python/reference/class/ApifyApiError
20. Dataset validation | Platform - Apify Documentation, acessado em junho 22, 2025, https://docs.apify.com/platform/actors/development/actor-definition/dataset-schema/validation
21. apify-client-python/src/apify_client/_errors.py at master - GitHub, acessado em junho 22, 2025, https://github.com/apify/apify-client-python/blob/master/src/apify_client/_errors.py
22. Build an Extraction Chain - LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/tutorials/extraction/
23. Structured Outputs from LLMs with LangChain - Opcito, acessado em junho 22, 2025, https://www.opcito.com/blogs/langchain-for-clean-object-based-responses-from-llmss
24. How to return structured data from a model | 🦜 LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/how_to/structured_output/
25. Natural Language Querying of GraphDB in LangChain - Ontotext, acessado em junho 22, 2025, https://www.ontotext.com/blog/natural-language-querying-of-graphdb-in-langchain/

26. Instagram Competitive Analysis: Key Insights and Strategies | Sprinklr, acessado em junho 22, 2025, https://www.sprinklr.com/blog/instagram-competitor-analysis/
27. Instagram engagement rate: How to calculate yours in 2025 - Sprout Social, acessado em junho 22, 2025, https://sproutsocial.com/insights/instagram-engagement-rate/
28. Instagram Engagement Rate in 2025 - Influencer Marketing Factory, acessado em junho 22, 2025, https://theinfluencermarketingfactory.com/instagram-engagement-rate/
29. Content Pillars for Social Media: Complete Guide for Small Business - Conversion Minded, acessado em junho 22, 2025, https://conversionminded.com/content-pillars/
30. Social Media Content Pillars: Boost Engagement and Consistency - Socialinsider, acessado em junho 22, 2025, https://www.socialinsider.io/blog/content-pillars-for-social-media/
31. Prompt engineering: A guide to improving LLM performance - CircleCI, acessado em junho 22, 2025, https://circleci.com/blog/prompt-engineering/
32. How to Personalize Digital Marketing Campaigns with LLM Prompts - Visualmodo, acessado em junho 22, 2025, https://visualmodo.com/how-to-personalize-digital-marketing-campaigns-with-llm-prompts/
33. Mastering Prompt Engineering for Marketing - Generative AI for Marketers - Trust Insights, acessado em junho 22, 2025, https://academy.trustinsights.ai/courses/mastering-prompt-engineering-marketing
34. 5 Timeless Prompt Engineering Principles for Reliable AI Outputs - General Assembly, acessado em junho 22, 2025, https://generalassemb.ly/blog/timeless-prompt-engineering-principles-improve-ai-output-reliability/
35. Ensuring Consistent JSON Output from LLMs on Amazon Bedrock - Community.aws, acessado em junho 22, 2025, https://community.aws/content/2wWabHxa0No1ZveOI7IMjQg6APP/ensuring-consistent-json-output-from-large-language-models-llms-on-amazon-bedrock
36. How to parse JSON output - LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/how_to/output_parser_json/
37. LangChain: Structured Outputs from LLM - Kaggle, acessado em junho 22, 2025, https://www.kaggle.com/code/ksmooi/langchain-structured-outputs-from-llm
38. Control LLM output with LangChain's structured and Pydantic output parsers - Atamel.Dev, acessado em junho 22, 2025, https://atamel.dev/posts/2024/12-09_control_llm_output_langchain_structured_pydantic/
39. How to use tools in a chain | 🦜 LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/how_to/tools_chain/
40. Tools | 🦜 LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/concepts/tools/

41. How to create tools | 🦜 LangChain, acessado em junho 22, 2025, https://python.langchain.com/docs/how_to/custom_tools/
42. Quickstart — python-docx 1.2.0 documentation - Read the Docs, acessado em junho 22, 2025, https://python-docx.readthedocs.io/en/latest/user/quickstart.html
43. Understanding Styles — python-docx 1.2.0 documentation, acessado em junho 22, 2025, https://python-docx.readthedocs.io/en/latest/user/styles-understanding.html
44. Cant apply table style in word doc using docx python - Stack Overflow, acessado em junho 22, 2025, https://stackoverflow.com/questions/31388536/cant-apply-table-style-in-word-doc-using-docx-python
45. python-docx — python-docx 1.2.0 documentation, acessado em junho 22, 2025, https://python-docx.readthedocs.io/
46. Working with Styles — python-docx 1.2.0 documentation, acessado em junho 22, 2025, https://python-docx.readthedocs.io/en/latest/user/styles-using.html
47. Working with Tables - Python .docx Module - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/python/working-with-tables-python-docx-module/
48. How do I change the font color in the cell under the table in the Word document?, acessado em junho 22, 2025, https://stackoverflow.com/questions/73876063/how-do-i-change-the-font-color-in-the-cell-under-the-table-in-the-word-document
49. python-docx Changing Table Cell Background Color. - YouTube, acessado em junho 22, 2025, https://www.youtube.com/watch?v=1Mgb95yigkk
50. [892] Change the background color of a table in a Word document - McDelfino - 博客园, acessado em junho 22, 2025, https://www.cnblogs.com/alex-bn-lee/p/17732468.html
51. python docx set table cell background and text color - Stack Overflow, acessado em junho 22, 2025, https://stackoverflow.com/questions/26752856/python-docx-set-table-cell-background-and-text-color
52. python-docx Documentation, acessado em junho 22, 2025, https://media.readthedocs.org/pdf/docstest/latest/docstest.pdf
53. How to make header rows for table bold using Python docx? - Stack Overflow, acessado em junho 22, 2025, https://stackoverflow.com/questions/55847686/how-to-make-header-rows-for-table-bold-using-python-docx
54. Data Visualization with Seaborn - Python - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/data-visualization/data-visualization-with-python-seaborn/
55. Python Seaborn Tutorial - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/python-seaborn-tutorial/
56. How to Save a Plot to a File Using Matplotlib? - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/python/how-to-save-a-plot-to-

a-file-using-matplotlib/
57. How to Save a Plot to a File Using Matplotlib? - GeeksforGeeks, acessado em junho 22, 2025, https://www.geeksforgeeks.org/how-to-save-a-plot-to-a-file-using-matplotlib/