Relatório de Processo ETL: Análise de Dados do Instagram

Vinícius de Paula R Carvalho

Baseado no projeto de 07 de agosto de 2025

Abstract

Este relatório detalha o processo de Extração, Transformação e Carga (ETL) implementado para coletar e processar dados de perfis do Instagram. O sistema foi desenvolvido para extrair informações de perfis, postagens e reels, limpá-las e consolidá-las em um formato estruturado para análise posterior. Todo o processo é orquestrado por um script principal em Python, utilizando a API da Apify para extração e a biblioteca Pandas para manipulação dos dados.

1 Introdução

O objetivo deste projeto é realizar a extração e limpeza de dados de fontes públicas do Instagram para análise de estratégias de comunicação digital. O processo foi automatizado através de um pipeline de ETL robusto, garantindo que os dados brutos sejam coletados, processados e armazenados de forma eficiente e estruturada.

O pipeline é dividido em três fases principais, conforme descrito neste documento:

1. **Extração:** Coleta de dados brutos de perfis, posts e reels utilizando um ator da plataforma Apify.
2. **Transformação:** Limpeza, reestruturação e enriquecimento dos dados extraídos para prepará-los para análise.
3. **Carga:** Armazenamento dos dados transformados em um arquivo Excel com múltiplas planilhas para fácil acesso e visualização.

2 Fase 1: Extração de Dados

A primeira etapa do processo consiste na extração de dados diretamente do Instagram. Esta fase é encapsulada na classe ‘Extract‘ do script ‘ETL.py‘.

2.1 Ferramentas e Configuração

A extração é realizada através da plataforma **Apify**, que permite a automação da coleta de dados da web. A comunicação com a API é feita pela biblioteca ‘apify_client‘ em Python.

A classe ‘Extract‘ é inicializada com os seguintes parâmetros:

- `apify_api_key`: Chave de API para autenticação na plataforma Apify.
- `links`: Uma lista de URLs de perfis do Instagram dos quais os dados serão extraídos.
- `resultsLimit`: Limite de resultados a serem extraídos (padrão: 30).

2.2 Processo de Extração

O processo é dividido em três métodos, cada um responsável por extrair um tipo diferente de dado: perfis, posts e reels.

2.2.1 Extração de Perfis

O método ‘extractProfiles‘ utiliza o ator ‘shu8hvrXbJbY3Eb9W‘ da Apify para buscar detalhes dos perfis especificados. Os resultados são salvos em um arquivo JSON chamado ‘profiles.json‘.

```
def extractProfiles(self):
    run_input = {
        "directUrls": self.links,
        "resultsType": "details",
        "searchType": "user"
    }
```

```
    run = self.client.actor("shu8hvrXbJbY3Eb9W").call(run_input=
    run_input)

    items = []
    for item in self.client.dataset(run["defaultDatasetId"]).
    iterate_items():
        items.append(item)

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

Listing 1: Código para extração de perfis - ETL.py

2.2.2 Extração de Posts e Reels

De forma similar, os métodos ‘extractPosts‘ e ‘extractReels‘ utilizam o mesmo ator, mas com configurações diferentes no ‘run_input‘ para buscar os posts e reels dos perfis. Os dados são salvos, respectivamente, nos arquivos ‘posts.json‘ e ‘reels.json‘.

3 Fase 2: Transformação de Dados

Após a extração, os dados brutos em formato JSON são lidos e processados. Esta fase é gerenciada pela classe ‘Transform‘ e é crucial para garantir a qualidade e a usabilidade dos dados.

3.1 Leitura e Preparação

Inicialmente, os três arquivos JSON (‘profiles.json‘, ‘posts.json‘, ‘reels.json‘) são lidos e carregados em DataFrames do Pandas, uma poderosa biblioteca para manipulação de dados em Python.

```
class Transform():
    def __init__(self):
        self.df_profiles = pd.read_json(settings.PROFILES_JSON)
        self.df_posts = pd.read_json(settings.POSTS_JSON)
        self.df_reels = pd.read_json(settings.REELS_JSON)
        # ... chamadas para m todos de transformacao
```

Listing 2: Leitura dos arquivos JSON - ETL.py

3.2 Limpeza e Estruturação

Cada DataFrame passa por um processo de limpeza e transformação específico:

- **Perfis (‘df_profiles‘):** Colunas desnecessárias são removidas e os tipos de dados são ajustados.
- **Posts (‘df_posts‘):** Tratamento semelhante ao dos perfis, com foco em manter apenas as colunas relevantes para a análise das postagens.
- **Reels (‘df_reels‘):** Além da limpeza de colunas, os comentários mais recentes (‘latestComments‘) são extraídos e armazenados em um DataFrame separado (‘df_latestComments‘). As duas fontes de postagens (posts e reels) são então unificadas em um único DataFrame (‘df_reels_posts‘) para facilitar a análise consolidada.

```python
def transformReels(self):
    # ... codigo de limpeza

    # Extrai comentarios
    self.df_latestComments = self.df_reels[['id', 'latestComments']].explode('latestComments').dropna()
    comments_normalized = pd.json_normalize(self.df_latestComments['latestComments'])
    self.df_latestComments = pd.concat([self.df_latestComments.reset_index(drop=True), comments_normalized], axis=1)

    # Unifica reels e posts
    self.df_reels_posts = pd.concat([
        self.df_reels.assign(type_post='reels'),
        self.df_posts.assign(type_post='posts')
    ])
```

Listing 3: Exemplo de transformação de reels - ETL.py

4 Fase 3: Carga dos Dados

A etapa final do processo é a carga dos DataFrames transformados em um destino final. A classe 'Load' é responsável por esta tarefa.

4.1 Armazenamento em Arquivo Excel

Os dados processados são salvos em um único arquivo Excel ('all_data.xlsx'), onde cada DataFrame é armazenado em uma planilha separada. Isso facilita a consulta e a análise exploratória dos dados por usuários finais.

As seguintes planilhas são criadas:

- `profiles`
- `posts`
- `reels`
- `reels_latestComments`
- `reels_posts` (dados unificados)

```python
class Load():
    def __init__(self, transformer: Transform):
        self.transformer = transformer
        self.getTransformedDataframes()
        self.loadDataframes()

    def loadDataframes(self):
        with pd.ExcelWriter(settings.ALL_XLSX, engine='openpyxl') as writer:
            self.df_profiles.to_excel(writer, sheet_name="profiles", index=False)
            self.df_posts.to_excel(writer, sheet_name="posts", index=False)
```

```
            self.df_reels.to_excel(writer, sheet_name="reels", index=
    False)
            self.df_latestComments.to_excel(writer, sheet_name="
    reels_latestComments", index=False)
            self.df_reels_posts.to_excel(writer, sheet_name="
    reels_posts", index=False)
```

Listing 4: Código para carregar os dados em Excel - ETL.py

5 Orquestração do Processo

O fluxo completo do ETL é orquestrado pela função 'ELT'. Esta função verifica se os arquivos JSON já existem. Se não existirem, ela executa a etapa de extração. Em seguida, prossegue com as etapas de transformação e carga.

```
def ELT(apify_api_key: str, links: list[str], results_limit: int = 30):
    files_exist = all([
        os.path.exists(settings.PROFILES_JSON),
        os.path.exists(settings.POSTS_JSON),
        os.path.exists(settings.REELS_JSON),
    ])

    if not files_exist:
        print("       [1/3] EXTRA  O: Arquivos n o encontrados,
    extraindo com Apify...")
        Extract(apify_api_key=apify_api_key, links=links, resultsLimit=
    results_limit)
        print("    [1/3] EXTRA  O: Conclu da.")

    else:
        print("    [1/3] EXTRA  O: Arquivos JSON encontrados, pulando
    a extra  o.")

    print("       [2/3] TRANSFORMA  O: Iniciando a transforma  o
    dos dados...")
    transformer = Transform()
    print("    [2/3] TRANSFORMA  O: Conclu da.")

    print("       [3/3] CARGA: Carregando os dados para o Excel...")
    Load(transformer)
    print("    [3/3] CARGA: Conclu da.")
    print(f"       Processo finalizado! Os dados est o dispon veis em
    : {settings.ALL_XLSX}")
```

Listing 5: Função principal de orquestração - ETL.py

6 Conclusão

O sistema de ETL descrito neste relatório automatiza com sucesso a coleta, limpeza e armazenamento de dados do Instagram. A arquitetura modular, dividida em classes para cada fase do processo, garante a manutenibilidade e escalabilidade do código. O resultado final é um conjunto de dados estruturado e pronto para ser utilizado em análises aprofundadas, como modelagem de linguagem natural e visualização de dados, conforme planejado nas próximas etapas do projeto.