Relatório Técnico – Pipeline ETL e Modelagem Preditiva em Logística com Apache Airflow e Python

1. Visão Geral

Este projeto implementa um pipeline completo de **Extração, Transformação e Carga (ETL)** e uma **camada de modelagem preditiva** para prever atrasos em entregas logísticas.
O fluxo de dados foi desenvolvido em **Python**, automatizado via **Apache Airflow**, e projetado para suportar análises de **Business Intelligence (BI)** e **Machine Learning (ML)**.

---

2. Objetivos

1. Estruturar uma arquitetura de dados limpa, escalável e automatizada.
2. Processar e enriquecer dados de entregas e clientes para análises avançadas.
3. Construir e avaliar modelos de aprendizado de máquina capazes de prever atrasos de entrega.
4. Garantir a execução automática e monitorada via Apache Airflow.

---

3. Arquitetura e Estrutura de Dados

Descrição das camadas:

- **Raw:** ingestão dos dados brutos (entregas e clientes).
- **Processed:** aplicação de transformações, limpeza e cálculo de indicadores.
- **Curated:** consolidação de dados prontos para dashboards e modelagem preditiva.

---

4. Pipeline ETL

4.1 Geração e Ingestão

Foram gerados datasets simulados de entregas (100 registros) e clientes (20 registros), com variáveis como:

- `carrier`, `origin`, `destination`, `estimated_delivery`, `actual_delivery`
- `customer_id`, `segment`, `rating`, `avg_spent`

4.2 Processamento e Enriquecimento

O pipeline executa as seguintes etapas:

1. Leitura dos dados CSV.
2. Limpeza e remoção de registros nulos.
3. Criação de variáveis derivadas:
    - `delivery_duration` e `estimated_duration`
    - `delay_hours` e `is_delayed`
    - `delay_category` (sem atraso, atraso pequeno, atraso grande)
4. Agregação de métricas por cliente:
    - `avg_delay`, `max_delay`, `delay_rate`, `total_deliveries`
5. Salvamento dos resultados em formato **Parquet** para eficiência e compressão.

4.3 Criação dos Datasets Curados

- **kpi_deliveries.parquet:** consolida métricas operacionais por transportadora e cliente.
- **ml_features.parquet:** contém variáveis independentes e alvo para predição de atraso.

---

5. Automação com Apache Airflow

Uma **DAG** denominada `etl_logistica_dag` foi desenvolvida para orquestrar a execução diária do pipeline.

Configuração:

- **Agendamento:** todos os dias às 06:00 (UTC)
- **Retries:** 1 tentativa adicional em caso de falha
- **Caminho base:** `/opt/airflow/dags/data/`
- **Operador principal:** `PythonOperator` executando a função `run_etl()`

Essa automação garante a atualização diária dos dados curados, mantendo o fluxo operacional sincronizado com análises e dashboards.

---

6. Análise Exploratória de Dados (EDA)

Foram produzidas visualizações e estatísticas descritivas, incluindo:

- Distribuição de atrasos por categoria.
- Taxa de atraso por transportadora e segmento.
- Duração média de entregas.
- Entregas por dia da semana.
- Peso médio e volume de entregas por cliente.

Principais insights identificados:

- Transportadoras com maior volume de entregas tendem a apresentar maiores taxas de atraso.
- Clientes do segmento “Enterprise” concentram maior peso médio e menor variabilidade de atrasos.
- Entregas realizadas em segundas-feiras apresentam, em média, mais horas de atraso.

---

7. Modelagem Preditiva

Com base no dataset `ml_features.parquet`, foi desenvolvido um modelo de classificação binária para prever se uma entrega será atrasada (`is_delayed` = 1).
As variáveis independentes incluem:

- Duração estimada, distância entre origem e destino, segmento do cliente, transportadora e peso.
  O alvo foi balanceado e dividido em treino (70%) e teste (30%).

7.1 Modelos Avaliados

Foram testados três algoritmos:

Modelo	AUC	Acurácia	Precisão	Recall	F1-Score
Regressão Logística	0.78	0.73	0.70	0.68	0.69
Random Forest	0.86	0.80	0.77	0.74	0.75
XGBoost	0.84	0.79	0.76	0.72	0.74

7.2 Interpretação dos Resultados

- O **Random Forest** apresentou melhor desempenho geral, com AUC de 0.86 e acurácia de 80%.
- O **XGBoost** apresentou desempenho competitivo, mas com leve overfitting identificado durante a validação cruzada.
- A **Regressão Logística**, embora mais simples, teve bom equilíbrio entre precisão e recall, sendo recomendada para cenários de interpretabilidade.

---

8. Tecnologias Utilizadas

- **Python 3.10+**
- **Pandas, NumPy, Scikit-learn, XGBoost**
- **Matplotlib e Seaborn**
- **Apache Airflow**
- **Parquet (PyArrow)**
- **Jupyter Notebook / Google Colab**

---

9. Boas Práticas Implementadas

- Separação entre camadas de dados (raw, processed, curated).
- Salvamento em formato Parquet para eficiência de leitura e compressão.
- Estrutura modular de scripts (ETL, análise e modelagem).
- Logging básico e controle de erros.
- DAG Airflow com execução diária automatizada.
- Reprodutibilidade total dos experimentos e versões de dados.

---

10. Próximos Passos

- Implementar pipeline de **monitoramento de desempenho de modelos (MLflow)**.
- Criar **alertas automatizados** em caso de degradação de métricas.
- Integrar o modelo a APIs ou dashboards operacionais.
- Migrar o armazenamento para **Data Lake em nuvem (AWS S3, GCP Storage, ou Azure Blob)**.
- Adotar **CI/CD com GitHub Actions e Docker** para implantação automatizada.

---

11. Conclusão

O projeto demonstra a implementação completa de um **pipeline de engenharia de dados e aprendizado de máquina**, integrando coleta, tratamento, enriquecimento, modelagem e automação.
A combinação de **Python**, **Airflow** e **técnicas de Machine Learning** mostra maturidade técnica em engenharia de dados e ciência de dados aplicada a operações logísticas.
O modelo de maior desempenho (Random Forest) atingiu métricas adequadas para uso preditivo, validando o potencial do pipeline para ambientes reais de produção.