VINÍCIUS HENRIQUE DOS SANTOS

MACHINE LEARNING ENGINEERING

BIG DATA PIPELINES

AULA 01

SUMÁRIO

O QUE VEM POR AÍ?

Boas-vindas à empolgante jornada pelo mundo dos data pipelines, a artéria vital das operações baseadas em dados que impulsionam o mundo moderno. Imaginem um mundo em que cada clique que você faz, cada transação que realiza ou até mesmo cada passo que você dá gera dados.

Estamos cercados por uma imensidão de dados e a capacidade de capturá-los, processá-los e transformá-los em insights acionáveis está revolucionando indústrias, da saúde à finança, do entretenimento à manufatura.

Mas como exatamente esses dados viajam do ponto A ao ponto B? Como eles são transformados de um fluxo bruto e caótico em informações claras e úteis? A resposta está nos data pipelines. Você entenderá como “pensar” em um data pipeline, como fazer as perguntas corretas e os desafios que te esperam no mundo real.

HANDS ON

Pense na empresa em que você trabalha ou gostaria de trabalhar. Imagine-se responsável por definir estratégias inteligentes para ajudar a alcançar os objetivos de negócio através dos data pipelines. Um baita desafio, não?

Para isso, você precisará entender quais são os pilares da construção de um pipeline de dados e quais são as perguntas que precisam ser feitas, pois essas respostas te ajudarão a definir as estratégias na hora de desenvolver cada detalhe dos seus programas.

SAIBA MAIS

ETL X ELT X LT

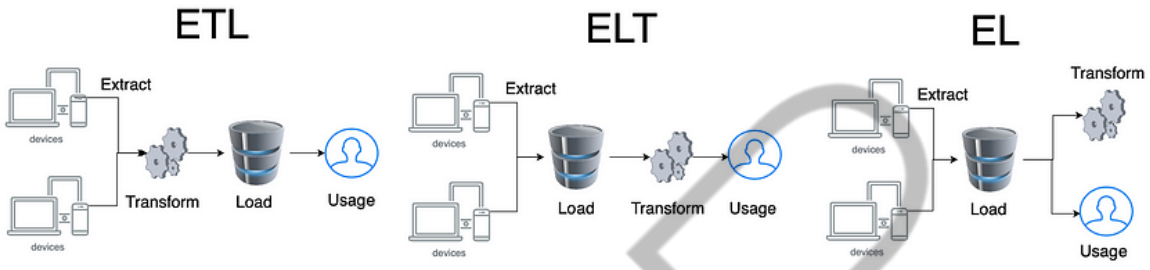

Figura 1 — ETL x ELT x EL
Fonte: Bolba (2023)

Os processos de engenharia de dados são cruciais para a efetiva integração e gestão de dados. ETL (Extração, Transformação, Carregamento), ELT (Extração, Carregamento, Transformação) e EL (Extração e Carregamento) são três abordagens comumente usadas nos fluxos de trabalho de engenharia de dados.

ETL envolve extrair dados de várias fontes, transformá-los e carregá-los em um sistema alvo. Já ELT foca em carregar primeiramente os dados brutos em um sistema alvo e depois realizar transformações dentro do sistema. EL, por outro lado, enfatiza o carregamento direto, sem transformações extensivas. Cada abordagem tem suas próprias vantagens e importância, baseadas nos requisitos específicos e características do processo de integração de dados. Exploraremos a importância do ETL, ELT e EL e forneceremos insights sobre sua criação e casos de uso na engenharia de dados.

ETL

ETL (Extração, Transformação, Carregamento) é um processo crucial na engenharia de dados, desempenhando um papel fundamental na integração e gestão de dados. Envolve extrair dados de várias fontes, transformá-los em um formato consistente e carregá-los em um sistema alvo ou armazém de dados. O ETL é

essencial para que as empresas utilizem efetivamente seus ativos de dados e tomem decisões informadas com base em informações precisas e consolidadas.

A importância do ETL na engenharia de dados é inegável. Ele permite que organizações combinem dados de fontes distintas, como bancos de dados, aplicações e arquivos, consolidando-os em um formato unificado. Ao extrair dados dessas fontes, ETL garante que insights valiosos possam ser derivados pela análise dos dados combinados.

A etapa de transformação do ETL envolve limpeza, validação e reestruturação dos dados para torná-los consistentes e utilizáveis. Esta etapa inclui tarefas como filtragem de dados, agregação, enriquecimento e conversões de tipos de dados. A transformação garante que os dados estejam em um formato adequado para análise, relatórios e outros processos subsequentes.

Uma vez que os dados são transformados, eles são carregados em um sistema alvo ou armazém de dados. Esse destino pode ser um banco de dados relacional, um data lake ou uma solução de armazenamento baseada na nuvem. Carregar os dados em um repositório central permite consultas, análises e relatórios eficientes, facilitando a extração de insights valiosos e a tomada de decisões de negócios.

Criar um processo de ETL envolve várias etapas. Primeiro, as fontes de dados precisam ser identificadas, incluindo bancos de dados, arquivos, APIs ou outros sistemas. O processo de extração envolve puxar os dados necessários dessas fontes usando várias técnicas, como consultas SQL, chamadas de API ou análise de arquivos.

Após a extração, os dados passam por transformação. Esta etapa inclui limpeza de dados para remover duplicatas, valores ausentes ou inconsistências. Checagens de validação de dados são realizadas para garantir a qualidade e a integridade dos dados. Tarefas de transformação de dados são implementadas usando linguagens de programação, scripts ou ferramentas ETL especializadas.

Os dados transformados são então carregados no sistema alvo ou armazém de dados. Esta etapa envolve mapear os dados transformados para o esquema de destino apropriado e usar técnicas de carregamento, como carregamento em massa ou atualizações incrementais.

ELT

ELT (Extração, Carregamento, Transformação) é uma abordagem alternativa na engenharia de dados que ganhou popularidade nos últimos anos. Diferentemente do processo tradicional de ETL, o ELT envolve extrair dados de várias fontes, carregá-los em um sistema alvo e, em seguida, realizar transformações diretamente dentro do sistema alvo. O ELT oferece várias vantagens e tornou-se um componente importante dos fluxos de trabalho modernos de engenharia de dados.

A importância do ELT na engenharia de dados reside na sua capacidade de alavancar o poder de processamento e escalabilidade de plataformas de dados modernas, como data lakes e soluções de armazenamento baseadas na nuvem. Ao carregar primeiro os dados brutos nessas plataformas e depois transformá-los usando frameworks de processamento distribuído, como Apache Spark, ou linguagens de processamento de dados, como SQL, as organizações podem aproveitar as capacidades de processamento paralelo e lidar com grandes volumes de dados de forma mais eficiente.

Criar um processo de ELT envolve várias etapas. Inicialmente, as fontes de dados são identificadas, semelhantemente ao processo de ETL. A fase de extração envolve a recuperação dos dados necessários dessas fontes usando métodos como consultas SQL, chamadas de API ou ingestão de arquivos. No entanto, diferentemente do ETL, os dados são carregados diretamente no sistema alvo, como um data lake ou uma solução de armazenamento baseada na nuvem, sem passar por transformações extensivas nesta etapa.

Uma vez que os dados são carregados, as transformações são realizadas dentro do sistema alvo. Isso normalmente envolve o uso de frameworks de processamento distribuído, consultas SQL ou linguagens de programação para transformar os dados em um formato consistente e utilizável. As transformações podem incluir limpeza de dados, agregação, junção e outras operações necessárias para análise e relatórios.

Um dos principais benefícios do ELT é a flexibilidade que oferece em termos de exploração e análise de dados. Ao carregar dados brutos no sistema alvo, engenheiros(as) de dados e analistas têm a liberdade de explorar e experimentar com diferentes transformações e análises, sem as limitações impostas por processos ETL

pré-definidos. O ELT permite uma engenharia de dados ágil, possibilitando que as organizações se adaptem rapidamente às mudanças nos requisitos de negócios e obtenham insights de maneira mais iterativa e exploratória. Ele é comumente usado em conjunto com data lakes ou casas de data lake.

EL

EL (Extração e Carregamento) é uma abordagem para engenharia de dados que se concentra em extrair dados de várias fontes e carregá-los diretamente em um sistema alvo, sem realizar transformações extensivas durante a fase de carregamento. O EL surgiu como uma alternativa viável ao tradicional processo de ETL (Extração, Transformação, Carregamento), oferecendo seu próprio conjunto de vantagens e importância nos fluxos de trabalho de engenharia de dados.

A importância do EL na engenharia de dados reside em sua habilidade de otimizar o processo de integração de dados e melhorar a eficiência. Ao separar as fases de extração e carregamento, o EL permite uma ingestão mais rápida de dados no sistema alvo, reduzindo o tempo necessário para tornar os dados disponíveis para análise e relatórios. Isso é particularmente útil ao lidar com grandes volumes de dados ou fluxos de dados em tempo real.

Criar um processo de EL envolve identificar as fontes de dados e determinar os métodos de extração. Várias técnicas podem ser empregadas, como consultar bancos de dados, usar APIs ou ingerir dados de arquivos. Os dados extraídos são então carregados no sistema alvo sem passar por transformações extensas. A fase de carregamento pode envolver carregamentos em massa diretos ou atualizações incrementais, dependendo da natureza dos dados e do sistema alvo.

O EL é particularmente adequado para cenários nos quais o sistema alvo possui capacidades internas para transformações e processamento de dados. Plataformas de dados modernas, como data lakes ou soluções de armazenamento baseadas na nuvem, frequentemente fornecem frameworks de processamento poderosos, como Apache Spark, ou motores de consulta distribuídos que podem realizar transformações nos dados carregados. Isso permite um processamento de dados flexível e escalável sem a necessidade de uma fase de transformação separada.

Embora o EL ofereça vantagens em termos de velocidade e flexibilidade, ele pode não ser adequado para todos os casos de uso. Transformações de dados complexas, limpeza de dados ou verificações de qualidade de dados são frequentemente melhor tratadas na abordagem tradicional de ETL. Portanto, é essencial avaliar cuidadosamente os requisitos e características dos dados antes de decidir adotar uma abordagem EL ou ETL.

Casos de uso e conclusão

A escolha entre ETL (Extração, Transformação, Carregamento), ELT (Extração, Carregamento, Transformação) ou EL (Extração e Carregamento) depende dos requisitos específicos e características do processo de integração de dados.

O ETL é comumente usado quando transformações de dados extensivas são necessárias antes de carregar os dados no sistema alvo. Esta abordagem é adequada quando as fontes de dados têm diferentes estruturas, formatos ou questões de qualidade de dados que precisam ser abordadas antes da análise. Ele permite limpeza, agregação e reestruturação dos dados durante a fase de transformação, garantindo a consistência e a integridade dos dados no sistema alvo.

O ELT é frequentemente preferido quando o sistema alvo possui capacidades internas para transformações e processamento de dados. Ao carregar primeiramente dados brutos no sistema alvo, ele aproveita o poder de processamento e escalabilidade de plataformas de dados modernas. Esta abordagem é benéfica em cenários nos quais as transformações de dados podem ser realizadas de forma eficiente dentro do sistema alvo, usando frameworks de processamento distribuídos ou motores de consulta.

Por fim, o EL é adequado quando o processo de integração de dados requer transformações mínimas ou nenhuma transformação de dados. É frequentemente usado em cenários nos quais as fontes de dados e o sistema alvo têm formatos e estruturas compatíveis, eliminando a necessidade de transformações extensivas. Ele foca em extrair dados de forma eficiente das fontes e carregá-los diretamente no sistema alvo, garantindo rápida disponibilidade para análise e relatórios.

A decisão de usar ETL, ELT ou EL depende de fatores como complexidade das transformações de dados, capacidades de processamento do sistema alvo, requisitos de qualidade de dados e o caso de uso específico em questão. É importante avaliar

cuidadosamente esses fatores e considerar os prós e contras de cada abordagem para determinar a solução mais apropriada para um determinado cenário de integração de dados.

Virtualização de dados

A virtualização de dados é uma tecnologia inovadora no campo da gestão de dados. Ela permite que os usuários acessem e manipulem dados sem a necessidade de conhecimento técnico sobre onde os dados estão fisicamente armazenados ou qual é sua estrutura.

Este conceito está se tornando cada vez mais popular à medida que as empresas lidam com volumes crescentes de dados provenientes de várias fontes. A ideia central é abstrair a camada física dos dados, permitindo que usuários se concentrem na análise e interpretação das informações para trazer vantagens competitivas para o negócio.

Os objetivos principais da virtualização de dados

- **Acesso unificado**: fornecer um ponto único de acesso a dados de várias fontes, facilitando a usuários a consulta e manipulação de informações sem se preocuparem com a localização ou formato dos dados.
- **Agilidade:** permitir que as organizações respondam rapidamente às mudanças nas necessidades de negócios, integrando novas fontes de dados sem grandes projetos de integração ou reestruturação. Em um primeiro momento, a virtualização de dados pode ser uma alternativa mais rápida do que a criação de um pipeline de dados mais complexo, trazendo a possibilidade de um MVP mais ágil até que o cenário “perfeito” com o pipeline esteja pronto.
- **Redução de custo:** diminuir os custos associados ao armazenamento e manutenção de grandes volumes de dados, uma vez que permite que os dados permaneçam em seus sistemas de origem.
- **Segurança e governança:** melhorar a segurança e a governança dos dados, fornecendo uma camada em que políticas de acesso e controle podem ser aplicadas de forma consistente através das próprias ferramentas de virtualização.

Quando usar:

- **Integração de dados de múltiplas fontes**: quando você precisa combinar dados de diferentes repositórios e formatos para análise e relatórios.
- **Acesso em tempo real**: quando o acesso em tempo real a informações atualizadas é crucial, como monitoramento de operações, análise financeira na bolsa de valores ou resposta a eventos de mercado.
- **Ambientes com mudanças frequentes**: em organizações nas quais os sistemas de dados e requisitos de negócios estão constantemente mudando, a virtualização oferece a flexibilidade necessária para se adaptar com mais flexibilidade do que em grandes pipelines.

Quando não usar:

- **Grandes volumes de dados para processamento intensivo:** se as operações requerem o processamento de grandes volumes de dados de forma intensiva, a virtualização pode introduzir uma latência inaceitável.
- **Ambientes com requisitos de segurança extremamente rígidos:** em alguns casos, a regulamentação ou políticas de segurança podem exigir que os dados sejam armazenados e processados de maneira específica, o que pode ser incompatível com a abordagem de virtualização.

Prós:

- **Flexibilidade e rapidez**: permite uma rápida integração de novas fontes de dados e a adaptação a mudanças nos requisitos de negócios.
- **Redução de custos**: menos necessidade de armazenamento e infraestrutura dedicada para a consolidação de dados.
- **Melhoria da qualidade e consistência dos dados**: oferece uma visão unificada dos dados, o que pode aumentar a consistência e a qualidade das informações trazendo os dados de diferentes silos para acessos unificados em um único “canal”.

Contras:

- **Desempenho:** pode haver problemas de desempenho, especialmente em sistemas que processam grandes volumes de dados ou requerem tempos de resposta muito rápidos.
- **Complexidade de gerenciamento**: a camada de virtualização adiciona uma nova peça ao ambiente de TI, o que pode aumentar a complexidade de gerenciamento e monitoramento.

Ferramentas

Várias ferramentas estão disponíveis no mercado para implementar a virtualização de dados, cada uma com suas próprias características e benefícios. Algumas das mais populares incluem:

- **Denodo**: oferece uma solução de virtualização de dados de alto desempenho com recursos avançados de otimização de consulta e gestão.
- **Informatica**: conhecida por suas soluções de integração de dados, a Informatica também oferece ferramentas robustas de virtualização de dados.
- **IBM Data Virtualization**: parte do portfólio da IBM, essa ferramenta oferece virtualização de dados integrada com outras soluções de dados e AI da empresa.

Casos de uso:

- **Business Intelligence (BI) e análise**: empresas usam virtualização de dados para fornecer acesso em tempo real a dados integrados de múltiplas fontes para ferramentas de BI e análise.
- **Migração de dados**: durante a migração de sistemas, a virtualização pode fornecer acesso contínuo a dados de sistemas legados e novos, minimizando a interrupção dos negócios.
- **Gerenciamento de dados do cliente**: unificar dados de clientes de várias fontes para fornecer uma visão completa do cliente, melhorando o atendimento e a personalização.

A virtualização de dados é uma abordagem poderosa para a gestão de dados moderna, oferecendo flexibilidade, agilidade e potencial de redução de custos e podendo fazer parte de um pipeline de dados mais robusto. Contudo, como qualquer tecnologia, não é uma solução universal e deve ser considerada dentro do contexto das necessidades específicas de negócios e dos desafios técnicos. Com a escolha da ferramenta certa e uma compreensão clara de quando e como implementar, a virtualização de dados pode ser um componente valioso na estratégia de dados de qualquer empresa.

Comparativo entre os tipos de arquivos

YAML, JSON, Parquet, Avro, CSV, Pickle e XML são exemplos de formatos de serialização de dados. Eles são comumente usados para representar dados de uma maneira estruturada que pode ser facilmente armazenada, enviada, recebida e processada por sistemas e aplicações. Cada um desses formatos possui seus pontos fortes e fracos e pode ser mais adequado para casos de uso específicos, dependendo de fatores como a complexidade dos dados, requisitos de desempenho e compatibilidade com outros sistemas. Confira um breve resumo sobre os principais tipos de arquivo:

YAML

YAML, abreviação de "YAML Ain't Markup Language", é um formato de serialização de dados legível por humanos frequentemente utilizado para arquivos de configuração, troca de dados e persistência. O YAML foi projetado para ser fácil de ler e escrever e visa ser uma alternativa mais amigável para humanos em comparação a outros formatos de serialização de dados, como JSON e XML.

No YAML, os dados são organizados em uma coleção de pares chave-valor, em que cada chave é uma string e cada valor pode ser uma string, número, booleano, array ou objeto. Os dados podem ser aninhados e indentados para criar uma hierarquia de valores e comentários podem ser adicionados para fornecer contexto ou documentação adicional. Aqui está um exemplo de como os dados podem ser armazenados no formato YAML.

```
import yaml

# Define o arquivo YAML como uma string Python
yaml_data = '''
- name: apple
  color: red
  weight: 0.25
  price:
    usd: 0.50
    eur: 0.42
  nutrients:
    - fiber
    - vitamin C
    - potassium

- name: banana
  color: yellow
  weight: 0.18
  price:
    usd: 0.30
    eur: 0.25
  nutrients:
    - fiber
    - vitamin B6
    - potassium
'''

# Carrega o YAML como uma string Python
data = yaml.load(yaml_data, Loader=yaml.FullLoader)

# Imprime na tela
print(data)

# Grava os dados em um arquivo YAML
with open('fruits.yaml', 'w') as f:
    yaml.dump(data, f)

# Lê o arquivo que foi gravado
with open('fruits.yaml', 'r') as f:
    data = yaml.load(f, Loader=yaml.FullLoader)
```

Código-fonte 1 – Exemplo de YAML
Fonte: Rayan Yassminh (2023)

JSON

JSON, abreviação de JavaScript Object Notation, é um formato leve e amplamente utilizado para transferência de dados. É um formato de texto fácil de ler e escrever, projetado para ser legível tanto por humanos quanto por máquinas. O

JSON é frequentemente usado para transmitir dados entre um servidor e uma aplicação web, assim como para armazenar dados em um formato de arquivo. Ele é baseado em uma estrutura de pares chave-valor, na qual os dados são organizados em objetos, arrays e valores.

Um objeto é delimitado por chaves “{}” e contém uma coleção de pares chave-valor, em que cada chave é uma string e cada valor pode ser uma string, número, booleano, array ou outro objeto. Um array é delimitado por colchetes “[]” e contém uma coleção de valores, em que cada valor pode ser uma string, número, booleano, array ou objeto. Um valor pode ser uma string, número, booleano, nulo, array ou objeto.

```
 import json

# Define o JSON como uma string Python
json_data = '''
[
  {
    "name": "apple",
    "color": "red",
    "weight": 0.25,
    "price": {
      "usd": 0.50,
      "eur": 0.42
    },
    "nutrients": [
      "fiber",
      "vitamin C",
      "potassium"
    ]
  },
  {
    "name": "banana",
    "color": "yellow",
    "weight": 0.18,
    "price": {
      "usd": 0.30,
      "eur": 0.25
    },
    "nutrients": [
      "fiber",
      "vitamin B6",
      "potassium"
    ]
  }
]
'''
```

```
# Carrega o JSON como um objeto Python
data = json.loads(json_data)

# Imprime na tela
print(data)

# Grava os dados em um arquivo JSON
with open('fruits.json', 'w') as f:
    json.dump(data, f)

# Lê o arquivo que foi gravado
with open('fruits.json', 'r') as f:
    data = json.load(f)
```

Código-fonte 2 – Exemplo de JSON
Fonte: Rayan Yassminh (2023)

Parquet

Parquet é um formato de armazenamento que armazena dados de maneira colunar. Os valores de uma única coluna são armazenados juntos, em oposição à abordagem tradicional de armazenamento linha a linha. Esse tipo de armazenamento pode permitir uma compressão mais eficiente e melhorar o desempenho das consultas.

O Parquet foi originalmente desenvolvido pela Cloudera e Twitter, como um esforço colaborativo, e agora é um padrão aberto comumente utilizado em ambientes de big data, como Apache Hadoop, Apache Spark ou EMR. O particionamento é uma característica importante do Parquet, permitindo que os dados sejam divididos em arquivos menores com base em colunas específicas.

Essa característica possibilita a execução eficiente de consultas e otimização do uso de memória. Além disso, o Parquet suporta tipos de dados complexos, como arrays e mapas, bem como estruturas aninhadas, como structs e arrays. Esses tipos de dados complexos podem ser achatados e armazenados de maneira compacta e eficiente, tornando o Parquet um formato adequado para armazenar dados hierárquicos.

```
import pyarrow.parquet as pq

# Define os dados como uma lista Python
```

```
    data = [     {'name': 'apple', 'color': 'red', 'weight':
0.25,  'price_usd':  0.50,  'price_eur':  0.42,  'nutrients':
['fiber', 'vitamin C', 'potassium']},
        {'name': 'banana', 'color': 'yellow', 'weight': 0.18,
'price_usd': 0.30, 'price_eur': 0.25, 'nutrients': ['fiber',
'vitamin B6', 'potassium']}
    ]

    # Converte a lista para um objeto (tabela) PyArrow
    table = pq.Table.from_pydict({k: [d.get(k, None) for d in
data] for k in data[0]})

    # Grava os dados como um Parquet
    pq.write_table(table, 'fruits.parquet')

    # Lê os dados do parquet
    table = pq.read_table('fruits.parquet')

    # Converte o objeto PyArrow para uma lista
    data    =    [{k:    v[i].as_py()    for    k,    v    in
table.to_pydict().items()} for i in range(table.num_rows)]

    # Imprime os dados
    print(data)
```

Código-fonte 3 – Exemplo de Parquet
Fonte: Rayan Yassminh (2023)

Avro

Avro é um sistema utilizado na serialização de dados para facilitar a troca de dados entre várias plataformas, linguagens de programação e sistemas. É um componente essencial do ecossistema Hadoop, desenvolvido pela Apache Software Foundation, e encontra ampla aplicação em sistemas de big data. O formato Avro é projetado para ser eficiente e compacto e utiliza uma abordagem baseada em esquema para descrever a estrutura dos dados. O Avro oferece suporte para codificação binária e compressão de dados, o que melhora significativamente sua eficiência em relação a outros formatos de serialização, como JSON e XML.

```
     import fastavro

    # Define o schema Avro
    schema = {
        "type": "record",
        "name": "fruit",
        "fields": [
            {"name": "name", "type": "string"},
```

```
            {"name": "color", "type": "string"},
            {"name": "weight", "type": "float"},
            {"name": "price_usd", "type": "float"},
            {"name": "price_eur", "type": "float"},
            {"name":  "nutrients",  "type":  {"type":  "array",
"items": "string"}}
        ]
    }

    # Define os dados como um dicionário Python
    data = [
        {"name":  "apple",  "color":  "red",  "weight":  0.25,
"price_usd":  0.50,  "price_eur":  0.42,  "nutrients":  ["fiber",
"vitamin C", "potassium"]},
        {"name": "banana", "color": "yellow", "weight": 0.18,
"price_usd":  0.30,  "price_eur":  0.25,  "nutrients":  ["fiber",
"vitamin B6", "potassium"]}
    ]

    # Grava os dados em um arquivo AVRO
    with open("fruits.avro", "wb") as avro_file:
        fastavro.writer(avro_file, schema, data)

    # Lê os dados do arquivo Avro
    with open("fruits.avro", "rb") as avro_file:
        reader = fastavro.reader(avro_file)
        data = [record for record in reader]

    # Imprime na tela
    print(data)
```

Código-fonte 4 – Exemplo de Avro
Fonte: Rayan Yassminh (2023)

CSV

CSV significa Valores Separados por Vírgula. É um formato de arquivo simples, usado para armazenar e trocar dados tabulares, como planilhas ou bancos de dados. Em um arquivo CSV, cada linha representa um registro e cada coluna representa um campo dentro desse registro. Os campos são separados por uma vírgula e cada linha é separada por um caractere de nova linha. Os arquivos CSV são amplamente utilizados porque são fáceis de criar, editar e analisar. Eles podem ser abertos pela maioria dos aplicativos de planilha, softwares de banco de dados e linguagens de programação. Arquivos CSV também são leves e eficientes, tornando-os adequados para grandes conjuntos de dados.

No entanto, arquivos CSV têm algumas limitações. Eles não podem armazenar estruturas de dados complexas, como objetos aninhados ou arrays, e não suportam tipagem de dados ou validação de esquema. Além disso, arquivos CSV podem ter problemas com codificação, caracteres de escape e tratamento de valores nulos ou vazios. No geral, CSV é um formato de arquivo popular e simples para armazenar e trocar dados tabulares. No entanto, pode não ser adequado para todos os casos de uso, especialmente ao lidar com estruturas de dados complexas ou exigir tipagem e validação de dados.

```
      import csv

     # Colocando os dados de frutas em objeto Python
     data = [     {'name': 'apple', 'color': 'red', 'weight':
0.25, 'price_usd': 0.50, 'price_eur': 0.42, 'nutrients': 'fiber,
vitamin C, potassium'},    {'name': 'banana', 'color': 'yellow',
'weight':   0.18,   'price_usd':   0.30,   'price_eur':   0.25,
'nutrients': 'fiber, vitamin B6, potassium'}]

     with open('fruits.csv', mode='w', newline='') as file:
         writer = csv.writer(file)
         writer.writerow(['Name',  'Color',  'Weight',  'Price
(USD)', 'Price (EUR)', 'Nutrients'])
         for fruit in data:
             writer.writerow([fruit['name'],    fruit['color'],
fruit['weight'],    fruit['price_usd'],    fruit['price_eur'],
fruit['nutrients']])

     # Lendo o arquivo CSV gravado
     with open('fruits.csv', mode='r') as file:
         reader = csv.DictReader(file)
         for row in reader:
             print(row)
```

Código-fonte 5 – Exemplo de CSV
Fonte: Rayan Yassminh (2023)

Neste exemplo, cada linha representa uma fruta, com colunas para o nome da fruta, cor e preço. A primeira linha contém os cabeçalhos das colunas, que indicam o nome de cada campo. Os valores em cada linha são separados por vírgulas e cada linha é separada por um caractere de nova linha.

Pickle

Pickle é um módulo Python que implementa protocolos binários para serializar e desserializar a estrutura de um objeto Python. O Pickle é específico do Python e não é adequado para comunicação entre diferentes linguagens de programação. Embora seja conveniente, não é recomendado para armazenamento de longo prazo ou transferência de dados entre sistemas devido a preocupações de segurança e compatibilidade.

```
import pickle

fruits = [
    {
        'name': 'apple',
        'color': 'red',
        'weight': 0.25,
        'price': {'usd': 0.50, 'eur': 0.42},
        'nutrients': ['fiber', 'vitamin C', 'potassium']
    },
    {
        'name': 'banana',
        'color': 'yellow',
        'weight': 0.18,
        'price': {'usd': 0.30, 'eur': 0.25},
        'nutrients': ['fiber', 'vitamin B6', 'potassium']
    }
]

# Grava o arquivo pickle
with open('fruits.pickle', 'wb') as f:
    pickle.dump(fruits, f)

# Lê o arquivo pickle
with open('fruits.pickle', 'rb') as f:
    loaded_fruits = pickle.load(f)
```

Código-fonte 6 – Exemplo de Pickle
Fonte: Rayan Yassminh (2023)

XML

XML (Extensible Markup Language) é uma linguagem de marcação amplamente utilizada para troca de dados entre diferentes sistemas, linguagens de programação e plataformas. É um formato padrão que é fácil de ler e interpretar tanto

por humanos quanto por máquinas. O XML permite que usuários definam suas próprias tags, o que o torna altamente personalizável e versátil.

O XML usa uma estrutura em árvore para representar os dados, com cada elemento representado por uma tag. Essas tags podem conter atributos, texto ou elementos filhos. As tags e atributos são definidos pelo usuário, o que facilita a representação de estruturas de dados complexas.

```
<?xml version="1.0" encoding="UTF-8"?>
<fruits>
  <fruit>
    <name>apple</name>
    <color>red</color>
    <weight>0.25</weight>
    <price>
      <usd>0.50</usd>
      <eur>0.42</eur>
    </price>
    <nutrients>
      <nutrient>fiber</nutrient>
      <nutrient>vitamin C</nutrient>
      <nutrient>potassium</nutrient>
    </nutrients>
  </fruit>
  <fruit>
    <name>banana</name>
    <color>yellow</color>
    <weight>0.18</weight>
    <price>
      <usd>0.30</usd>
      <eur>0.25</eur>
    </price>
    <nutrients>
      <nutrient>fiber</nutrient>
      <nutrient>vitamin B6</nutrient>
      <nutrient>potassium</nutrient>
    </nutrients>
  </fruit>
</fruits>
```

Código-fonte 7 – Exemplo de XML
Fonte: Rayan Yassminh (2023)

Tabela resumo:

Formato	extenção do arquivo	Lido por humanos?	Suporta compressão?	Suporta esquemas?	Orientação da gravação	Uso
YAML	.yaml, .yml	SIM	SIM	SIM	Documento	Armazenamento de dados estruturados, arquivos de configuração, comunicação entre aplicações
JSON	.json	SIM	NÃO	NÃO	Objeto	Envio e recebimento de dados na internet, serviços WEB, Bancos de dados NoSQL
Parquet	.parquet	NÃO	SIM	SIM	Colunar	Processamento de dados em ambiente big data, data wherehousing, aplicações de analytics
Avro	.avro	NÃO	SIM	SIM	Objeto	Processamento de dados em ambiente big data, data wherehousing, aplicações de analytics
CSV	.csv	SIM	NÃO	NÃO	Linha	Aplicativos de planilhas, aplicações de bancos de dados, analise de dados
Pickle	.pkl	NÃO	NÃO	NÃO	Objeto	Uso especifico dentro de estruturas Python, serialização de objetos, comunicação entre processos
XML	.xml	SIM	SIM	SIM	Documento	Serviços WEB, geração de documentos, arquivos de configuração

Figura 2 — Tabela de resumo
Fonte: elaborado pelo autor (2024)

Comparativo de tamanho/processamento:

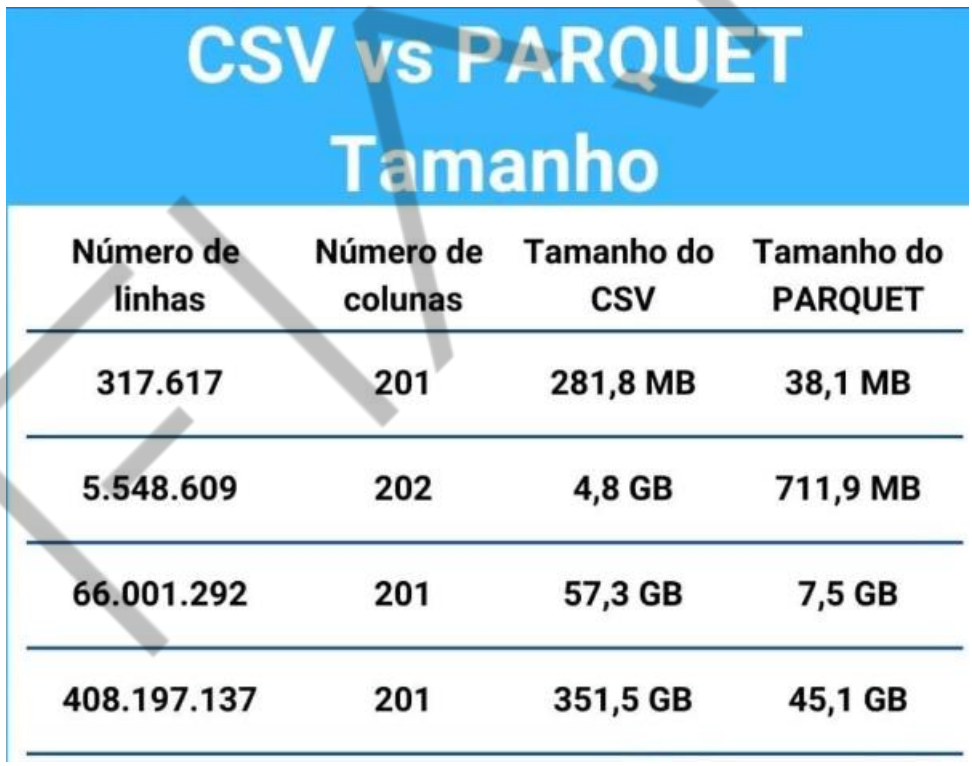

Figura 3 — CSV vs PARQUET
Fonte: elaborado pelo autor (2024)

O QUE VOCÊ VIU NESTA AULA?

Nessa aula você entendeu conceitualmente o que é um data pipeline, foi apresentado(a) para alguns componentes que fazem parte do ecossistema de transferência e transformação dos dados, entendeu quais são os pilares que precisam ser pensados antes de iniciar o desenvolvimento de um pipeline e entendeu quais são as perguntas que devem ser feitas e as respostas que te ajudarão a entregar um melhor produto final para a empresa.

Além de tudo isso, conseguiu ter uma ideia dos desafios existentes no mercado e algumas características e desafios enfrentados por algumas empresas na pesquisa que analisamos juntos.

REFERÊNCIAS

AMAZON. **What is XML?** 2023. Disponível em: <https://aws.amazon.com/what-is/xml/#:~:text=An%20Extensible%20Markup%20Language%20(XML,like%20Notepad%20or%20Notepad%2B%2B>. Acesso em: 28 mai. 2024.

APACHE. **Documentation**. 2023. Disponível em: <https://avro.apache.org/docs/>. Acesso em: 28 mai. 2024.

BOLBA, C. **Concepts for Data Engineers: ETL x ELT x EL**. Disponível em: <https://cassio-bolba.medium.com/concepts-for-data-engineers-etl-x-elt-x-el-7bed93bd9e45>. Acesso em: 28 mai. 2024.

HOUSLEY, M. **Fundamentals of Data Engineering: Plan and Build Robust Data Systems.** Sebastopol: O'Reilly Media, 2022.

JSON.ORG. **Introdução ao JSON**. 1999. Disponível em: <https://www.json.org/json-pt.html>. Acesso em: 28 mai. 2024.

PARQUET. **Documentation**. 2023. Disponível em: <https://parquet.apache.org/docs/>. Acesso em: 28 mai. 2024.

REDHAT. **O que é YAML?**. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/automation/what-is-yaml#:~:text=O%20YAML%20%C3%A9%20uma%20linguagem,marca%C3%A7%C3%A3o)%20%5Bacr%C3%B4nimo%20recorrente%5D>. Acesso em: 28 mai. 2024.

REIS, J. **Fundamentals of Data Engineering: Plan and Build Robust Data Systems.** Sebastopol: O'Reilly Media, 2022.

VIRTUALITY, Data. **Data Virtualization: The Complete Overview**. ano da publicação. Disponível em: < https://datavirtuality.com/en/blog/data-virtualization-the-complete-overview/ > Acesso em: 28 mai. 2024.

YASSMINH, R. **What Are the Differences Between Data Serialization Formats: YAML, JSON, Parquet, Avro, CSV, Pickle, and XML?**. Disponível em: <https://medium.com/@ryassminh/what-are-the-differences-between-data-serialization-formats-yaml-json-parquet-avro-csv-9feb8ae50122>. Acesso em: 28 mai. 2024.

PALAVRAS-CHAVE

Palavras-chave: Pipeline. ETL. ELT. LT. Dados. Fluxo. AirFlow. Batch. Stream. Spark. PySpark.