Licenciatura em Engenharia de sistemas Informáticos

Integração de Sistemas de Informação

Relatório do Trabalho Prático

Renato Barbosa 22570

Professor: Óscar Ribeiro

Outubro de 2025

Índice

Índice de figuras

Introdução

Este trabalho tem como objetivo desenvolver e documentar um processo ETL (Extract, Transform, Load) utilizando a plataforma KNIME, aplicado a um cenário fictício de vendas de uma loja online. O projeto aborda a integração de dados provenientes de diferentes fontes e formatos, nomeadamente ficheiros CSV, XML e uma API em formato JSON.

Com este caso prático, pretende-se demonstrar como o KNIME pode ser usado para extrair, transformar e carregar dados de forma automatizada, assegurando a limpeza, normalização e integração das informações. O resultado final inclui a exportação dos dados em vários formatos e o armazenamento numa base de dados PostgreSQL, refletindo um processo completo de integração de sistemas de informação.

.

Problema

O problema consiste em consolidar dados de diferentes origens, clientes em CSV, produtos obtidos por API em JSON e vendas em XML num único conjunto de informação coerente e pronto para análise.

Durante o processo, foi também essencial aplicar transformações de normalização e validação dos dados, garantindo que todos os registos fossem tratados de forma uniforme e compatível. Foram calculadas métricas de negócio relevantes, como o total gasto por cliente, o total de vendas por produto e o total de vendas por categoria, permitindo obter uma visão global do desempenho comercial da loja.

O processo termina com a exportação dos resultados em vários formatos e o carregamento na base de dados PostgreSQL, assegurando a centralização e integridade da informação.

Estratégia Utilizada

Tratamento de Dados API

A primeira fonte de dados correspondeu à API utilizada para recolher as informações sobre os produtos da loja.

Através do node GET Request, foi feito o pedido ao endpoint, obtendo-se os dados em formato JSON. Com o node JSON Path, foram selecionados apenas os campos relevantes, nomeadamente o identificador do produto, o nome, a categoria e o preço. O node Ungroup converteu os arrays do JSON em linhas individuais, e com o JSON to Table os dados foram transformados em formato tabular. Por fim, o Column Renamer ajustou os nomes das colunas para facilitar a integração posterior. O resultado foi uma tabela de produtos limpa e organizada, pronta para ser combinada com as restantes fontes.

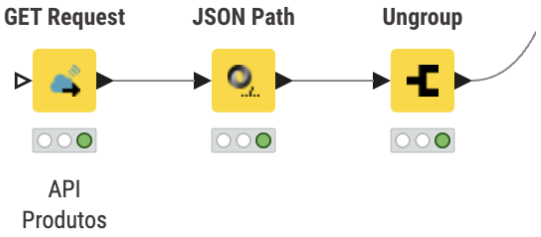

Figura 1 - Nodes utilizados para recolha de informação da API

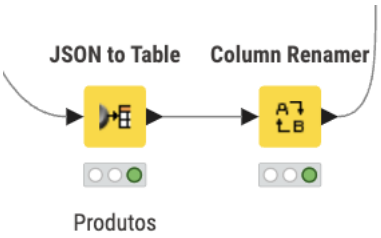

Figura 2 - Nodes utilizados na transformação dos dados da API

Tratamento de Dados CSV

A segunda fonte de dados correspondeu a informação de clientes, guardada num ficheiro CSV. A importação foi realizada através do node CSV Reader, a partir do qual foram aplicadas várias transformações para normalizar e validar a informação. Os nodes String Manipulation foram utilizados para uniformizar os nomes e as cidades, garantindo que todas as letras iniciais surgissem em maiúsculas. Posteriormente, o node Expression verificou o tamanho dos números de telefone e o Rule Engine foi usado para remover os registos cujos telefones não possuíam nove dígitos válidos. Foram ainda efetuadas correções aos emails com nodes Expression e String Manipulation, que trataram de erros nos domínios e adicionaram o sufixo ".com" quando em falta. Como resultado, obteve-se uma tabela de clientes limpa, sem valores inconsistentes e com todos os campos normalizados.

Figura 3 - Nodes utilizados para recolha de informação CSV

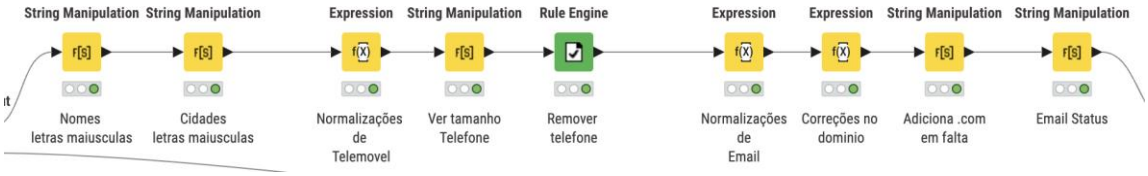

Figura 4 - Nodes utilizados para a normalização da informação CSV

Tratamento de Dados XML

A terceira fonte de dados correspondeu ao ficheiro XML que contém o histórico das vendas. A leitura foi realizada com o node XML Reader e o processamento seguinte com o XPath, que permitiu extrair os campos principais: identificador da venda, identificador do cliente, identificador do produto, quantidade e data. Em seguida, o node String to Date&Time converteu o campo de data para o formato apropriado, enquanto o String to Number transformou as colunas numéricas, assegurando que os valores pudessem ser corretamente utilizados em cálculos e junções. O resultado foi uma tabela de vendas consistente, com os tipos de dados devidamente preparados para integração.

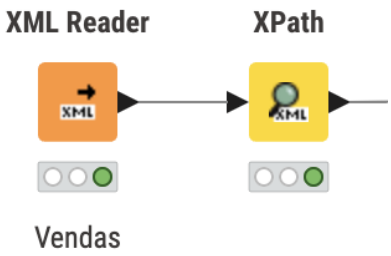

Figura 5 - Nodes utilizados para recolha de informação XML

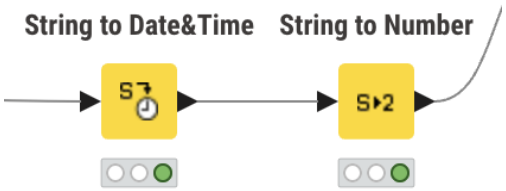

Figura 6 - Nodes utilizados para a normalização da informação XML

Integração dos dados

Com todas as fontes normalizadas, foi realizada a integração através de dois nodes Joiner. O primeiro juntou os dados das vendas com os clientes, utilizando o campo ClienteID, e o segundo ligou o resultado aos produtos através do campo ProdutoID. Depois de integrados, os dados passaram pelo node Math Formula, onde foi calculado o valor total de cada venda através da multiplicação da quantidade pelo preço. Os nodes Column Resorter e Sorter garantiram a organização final das colunas e a ordenação lógica dos resultados, produzindo uma tabela consolidada com toda a informação relevante sobre clientes, produtos e vendas.

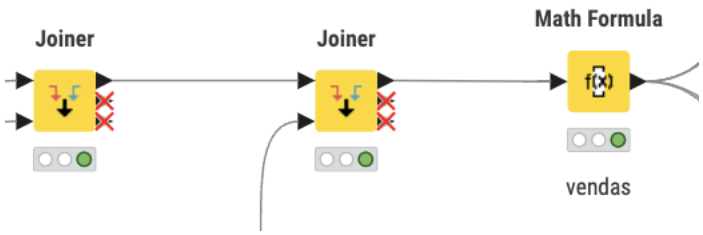

Figura 7 - Integração dos dados 1

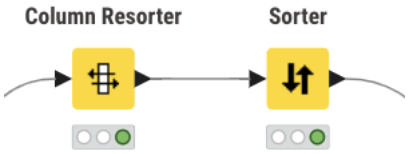

Figura 8 - Integração dos dados 2

Exportação e Armazenamento dos Resultados

Após a integração, os dados foram preparados para exportação em diferentes formatos, com o objetivo de assegurar a disponibilidade da informação em vários contextos de utilização. Os resultados foram exportados em ficheiros CSV, Excel e JSON, permitindo a partilha e análise em múltiplas plataformas. Paralelamente, foi estabelecida uma ligação à base de dados PostgreSQL, onde a tabela final foi gravada através do node DB Writer, garantindo a persistência e centralização dos dados processados. Estas operações completam o ciclo ETL, assegurando que o processo termina com a informação devidamente tratada, armazenada e pronta para consulta ou visualização.

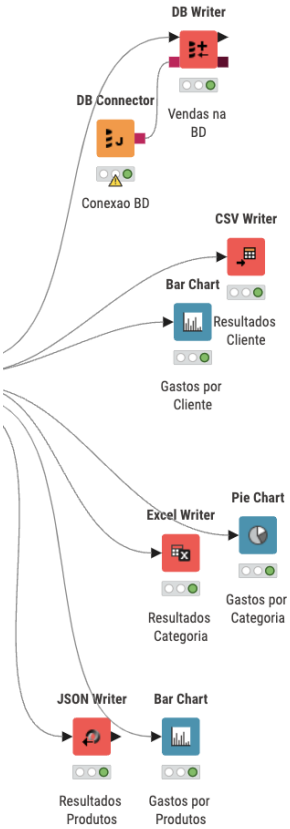

Figura 9 - Exportação e armazenamento de resultados

Transformações

Dados do CSV (Clientes)

A primeira fonte de dados correspondeu à base de clientes, guardada num ficheiro CSV. O tratamento foi dividido em várias etapas, cada uma com o seu propósito específico de normalização, validação e limpeza dos registos.

1. **CSV Reader** :
   Este node é responsável por importar o ficheiro **clientes.csv** e criar a tabela inicial com todos os campos. Foi garantindo que todas as colunas fossem corretamente reconhecidas pelo KNIME.

2. **String Manipulation**:
   O primeiro node desta categoria teve como objetivo uniformizar os nomes dos clientes. A função utilizada, **capitalize($Nome$)**, converte a primeira letra de cada nome em maiúscula, assegurando consistência na apresentação.

3. **String Manipulation**:
   O segundo node aplicou o mesmo princípio à coluna Cidade, com a expressão **capitalize($Cidade$)**, de forma a normalizar a grafia das localidades

4. **Expression**:
   Nesta etapa foram criadas quatro expressões dedicadas à limpeza e padronização dos números de telefone. As operações incluíram a substituição de letras por dígitos (**l** por **1** e **O** por **0**), a remoção de caracteres não numéricos e a eliminação de prefixos internacionais como **+351, 00351** ou **351**. O conjunto de expressões garantiu que apenas permanecessem números válidos no formato nacional.

5. **String Manipulation**:
   Este node criou uma nova coluna denominada **Tamanho_Telefone**, que mede o comprimento de cada número de telefone com a função **length($Telefone$)**. Esta informação serviu de base para o controlo de validade aplicado no passo seguinte.

6. **Rule Engine**:
   Através deste node foram removidos ou anulados os registos que não possuíam telefones com nove dígitos válidos. A regra aplicada foi **$Tamanho_Telefone$ = 9 => $Telefone$**, sendo que todos os restantes casos foram substituídos por valores vazios.

7. **Expression**:
   Este node tratou da normalização dos endereços de email, aplicando um conjunto de seis expressões que removem espaços, pontos e traços redundantes, eliminam caracteres inválidos e convertem todas as letras para minúsculas. O resultado foi uma estrutura limpa e consistente dos endereços eletrónicos.

8. **Expression**:
   Nesta etapa foram aplicadas regras adicionais de correção de domínios, substituindo terminações incorretas como .**con**, **.cmo** ou .**cm** por **.com** e corrigindo domínios comuns como **hotnail.com** para **hotmail.com** e **outlok.com** para **outlook.com**.

9. **String Manipulation**:
   Este node foi utilizado para adicionar automaticamente o sufixo ".com" em emails de provedores conhecidos que não o possuíssem, recorrendo à expressão
   regexReplace($Email$,"(?i)(@gmail|@hotmail|@outlook|@yahoo)$", "$1.com").

10. **String Manipulation**:
    Por fim, foi criada uma coluna de estado que valida se o formato do email é válido ou não. A expressão
    regexMatcher($Email$, "^(?!.*\\.\\.)[A-Za-z0-9]+([._%+-]?[A-Za-z0-9]+)*@[A-Za-z0-9-]+(\\.[A-Za-z]{2,})+$")
    retorna verdadeiro para endereços válidos e falso para inválidos, permitindo identificar facilmente possíveis erros residuais.

Dados da API (Produtos)

A segunda fonte de dados correspondeu à informação proveniente de uma API pública, utilizada para recolher a lista de produtos da loja online. A API em questão encontra-se disponível em https://fakestoreapi.com/products e fornece dados fictícios de comércio eletrónico, nomeadamente identificadores, nomes, categorias e preços de produtos.

1. **GET Request**:
   Este node foi responsável por aceder ao endpoint da API e realizar o pedido HTTP, retornando a informação em formato JSON. O ficheiro obtido continha um conjunto de objetos representando os produtos disponíveis, incluindo os campos **id**, **title**, **price** e **category**.

2. **JSON Path**:
   Após o carregamento, este node foi utilizado para extrair apenas os campos relevantes do JSON, evitando a importação de dados supérfluos. As expressões definidas permitiram selecionar o identificador, o nome, o preço e a categoria de cada produto.

3. **Ungroup**:
   Este node foi aplicado para desagrupar os arrays resultantes da extração, convertendo cada objeto JSON num registo independente. Desta forma, cada produto passou a ocupar uma linha própria na tabela, facilitando as transformações seguintes.

4. **JSON to Table**:
   A estrutura JSON foi convertida para o formato tabular padrão do KNIME, tornando os dados compatíveis com os restantes fluxos do processo ETL.

5. **Column Renamer**:
   Por fim, as colunas foram renomeadas para manter uma nomenclatura uniforme em todo o projeto. As alterações aplicadas foram: **id** para **ProdutoID**, **title** para **NomeProduto**, **price** para **Preço** e **category** para **Categoria**.

Dados do XML (Vendas)

A terceira fonte de dados correspondeu ao ficheiro **vendas.xml**, utilizado para representar o histórico das transações realizadas na loja online. Este conjunto contém informações sobre o identificador da venda, o cliente associado, o produto adquirido, a quantidade e a data correspondente a cada operação.

1. **XML Reader**:
   Este node foi responsável por importar o ficheiro **vendas.xml** para o KNIME, convertendo o conteúdo XML num formato compreensível para o sistema.

2. **XPath**:
   Após a leitura, o node XPath foi utilizado para extrair os campos relevantes do documento XML. Foram selecionados os elementos **VendaID**, **ClienteID**, **ProdutoID**, **Quantidade** e **Data**, transformando cada conjunto de tags numa linha independente. Este processo converteu a estrutura XML numa tabela tabular organizada e facilmente manipulável.

3. **String to Date&Time**:
   O campo **Data**, inicialmente armazenado como texto, foi convertido para o tipo de dados de data através deste node. Essa transformação permitiu realizar ordenações e análises temporais, garantindo a consistência de formato entre todos os registos.

4. **String to Number**:
   Este node foi aplicado para converter os campos numéricos (**VendaID**, **ClienteID**, **ProdutoID** e **Quantidade**) que se encontravam em formato texto para valores numéricos reais. Esta conversão assegurou a compatibilidade dos dados durante os cálculos e nas junções com as restantes tabelas do projeto.

Integração e cálculos

Com as três fontes de dados devidamente tratadas e validadas, foi realizada a fase de integração, onde se juntam os registos de clientes, produtos e vendas num único conjunto de dados. Nesta etapa, além das junções entre tabelas, foram também aplicados cálculos e operações de agregação que permitiram gerar métricas de negócio e preparar os resultados finais para exportação.

1. **Joiner**:
   O primeiro node Joiner foi utilizado para unir a tabela de clientes com a de vendas, estabelecendo a ligação através do campo **ClienteID**. Este passo permitiu associar a cada venda as respetivas informações do cliente, garantindo a coerência entre as duas fontes.

2. **Joiner**:
   O segundo node Joiner ligou o resultado anterior à tabela de produtos, tendo como chave o campo **ProdutoID**. Após esta operação, cada linha passou a conter todos os dados necessários, cliente, produto, quantidade, preço e data, criando a base de informação completa da loja online.

3. **Math Formula**:
   Este node foi aplicado para calcular o valor total de cada venda, utilizando a expressão **$Quantidade$ * $Preço$**. O resultado foi armazenado numa nova coluna denominada **TotalVenda**, representando o montante gasto em cada transação individual. Este valor foi posteriormente utilizado para gerar os diferentes relatórios e análises gráficas.

A partir deste ponto, o processo dividiu-se em quatro caminhos: um para a gravação da tabela completa e três para os relatórios agregados, que calculam os totais por cliente, categoria e produto.

4. **Column Resorter**:
   A partir do resultado do cálculo, foi utilizado este node para reorganizar a ordem das colunas, dispondo os campos de forma lógica e consistente (identificadores, informações do cliente e produto, e métricas no final). Esta organização torna a leitura dos dados mais clara e facilita o carregamento na base de dados.

5. **Sorter**:
   Em seguida, foi aplicado um Sorter que ordena a tabela resultante por nome do cliente, em ordem alfabética de A a Z. Esta ordenação garante uma disposição uniforme e facilita a verificação visual dos dados antes da gravação final na base de dados PostgreSQL.

6. **GroupBy**:
   Paralelamente, foram criadas três agregações distintas a partir do resultado do Math Formula. A primeira agrupou os dados por cliente, somando o total gasto por cada um. Esta operação permitiu identificar os clientes com maior volume de compras e serviu de base para os relatórios individuais.

7. **Sorter**:
   O resultado do GroupBy anterior foi ordenado por valor total gasto em ordem decrescente, destacando os clientes com maior despesa no topo do ranking.

8. **GroupBy**:
   A segunda agregação agrupou os dados por categoria de produto, calculando o total de vendas em cada categoria. Esta análise permitiu observar o desempenho de cada área da loja, como informática, escritório ou redes.

9. **Sorter**:
   O resultado foi novamente ordenado por valor de vendas em ordem decrescente, facilitando a interpretação gráfica e textual dos resultados.

10. **GroupBy**:
    A terceira agregação foi feita por produto, somando o total de vendas individuais de cada artigo. Desta forma, foi possível determinar quais os produtos mais vendidos e o respetivo peso no total das receitas.

11. **Sorter**:
    Este Sorter final ordenou os produtos pelo valor de vendas mais alto, permitindo uma visão rápida dos artigos com melhor desempenho comercial.

12. **Table to JSON**:
    Finalmente, os resultados consolidados foram convertidos para formato JSON através deste node, preparando a informação para exportação em ficheiros estruturados ou para integração com outras aplicações.

Exportações e visualização

Após o cálculo e a agregação dos resultados, o fluxo foi dividido em quatro saídas principais, cada uma com uma finalidade específica. A primeira corresponde à gravação da tabela consolidada na base de dados PostgreSQL através dos nodes **DB Connector** e **DB Writer**. As restantes três saídas foram utilizadas para a criação dos relatórios de análise: um ficheiro **CSV** com o total gasto por cliente acompanhado de um gráfico de barras, um ficheiro **Excel** com o total por categoria associado a um gráfico de pizza, e um ficheiro **JSON** com o total por produto representado num gráfico de barras.

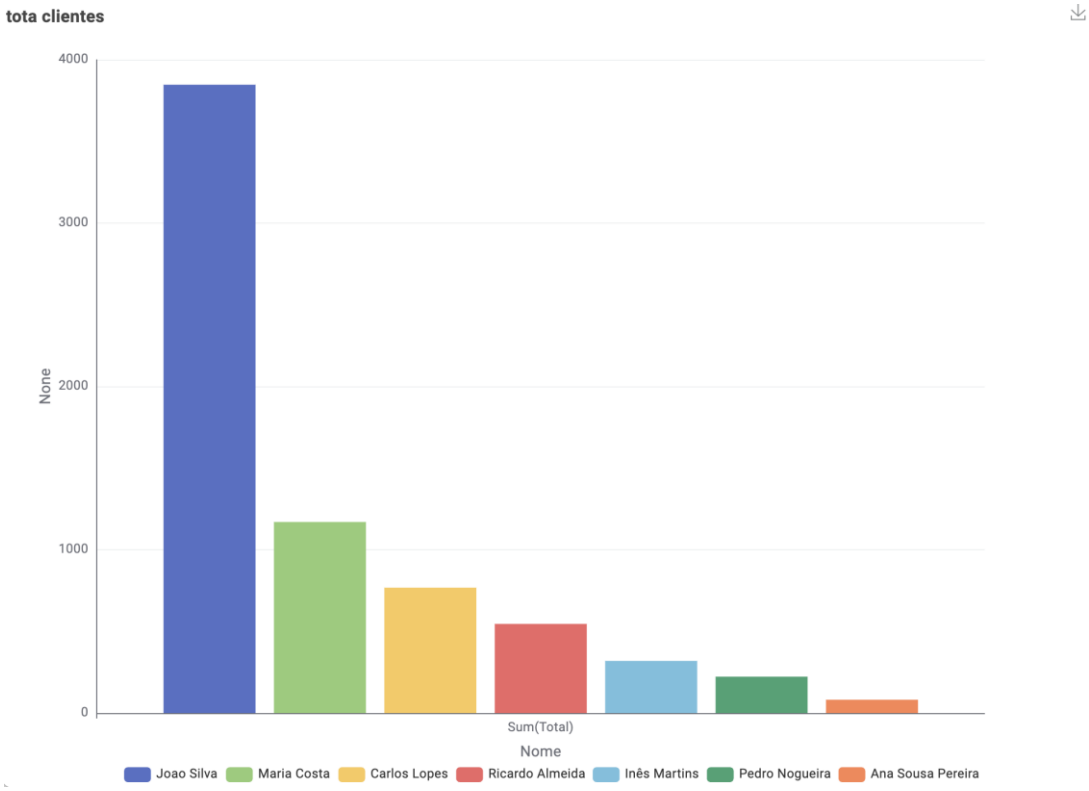

Figura 10 - Total gasto por cliente

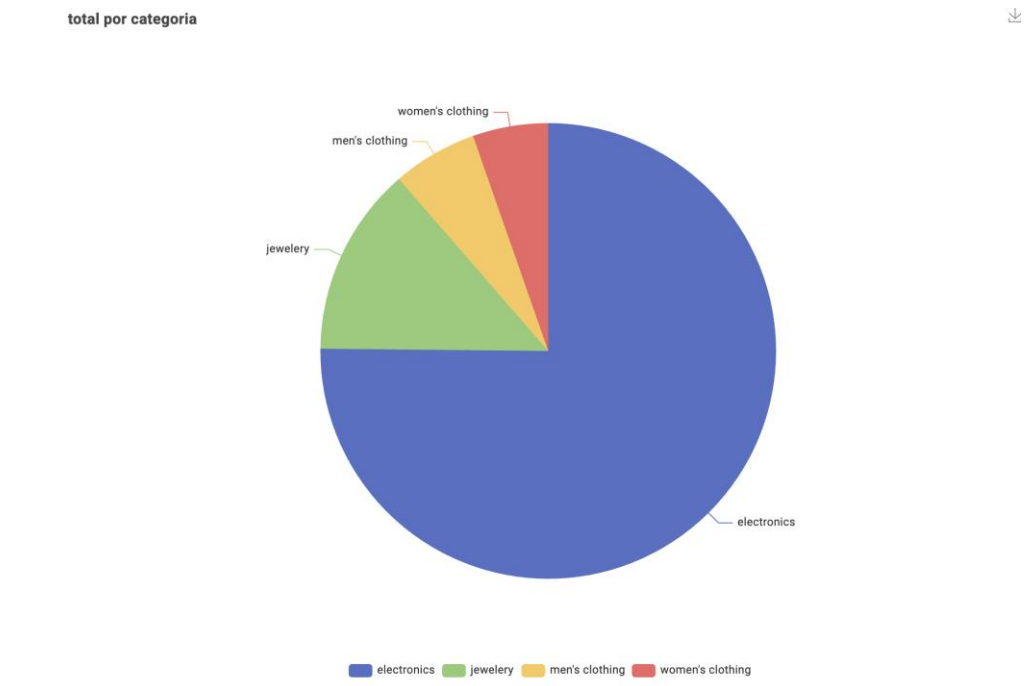

Figura 11 - Total gasto por categoria

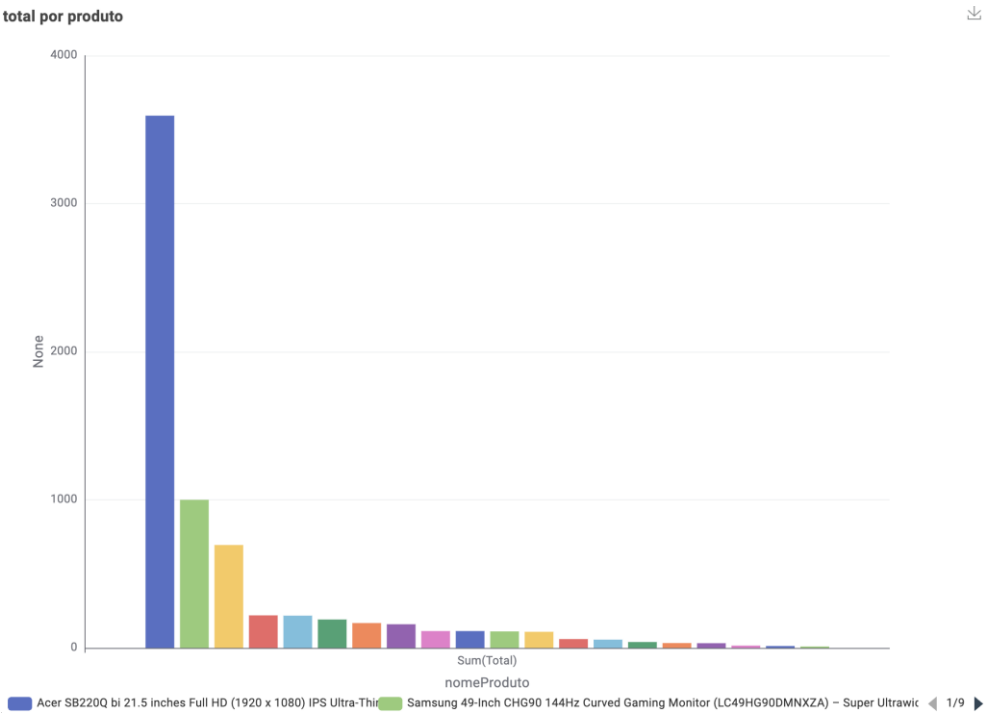

Figura 12 - Total gasto por produto

Jobs

O projeto foi estruturado em quatro componentes principais, correspondentes às etapas fundamentais do processo ETL. Esta divisão permite uma organização mais clara do fluxo, facilitando a execução e manutenção do projeto.

1. **Componente "Extract"**:
   Responsável pela recolha dos dados das diferentes fontes. Aqui são lidos o ficheiro **clientes.csv**, o ficheiro **vendas.xml** e a informação obtida através da **API**. Este componente garante que todos os dados são carregados corretamente no início do processo.

2. **Componente "Normalizações"**:
   Este componente realiza as operações de limpeza e preparação dos dados. Inclui as normalizações dos nomes, cidades, telefones e emails dos clientes. O objetivo é assegurar a consistência e integridade de todos os registos antes da integração.

3. **Componente "Transform"**:
   Nesta fase ocorrem as operações de junção e cálculo. São utilizadas duas junções para combinar as tabelas de clientes, vendas e produtos, seguidas do cálculo do valor total de cada venda. Também são gerados os agrupamentos de totais por cliente, categoria e produto, que alimentam as visualizações e relatórios.

4. **Componente "Load"**:
   O último componente trata da exportação dos resultados. Aqui são gerados os ficheiros em formato CSV, Excel e JSON, criados os gráficos de análise e efetuada a gravação final da tabela consolidada na base de dados PostgreSQL.

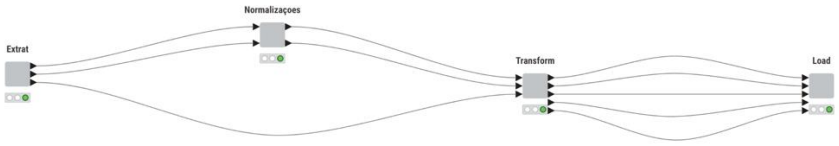

Figura 13 - Componentes KNIME

Vídeo com demonstração (QR Code)

Conclusão

A realização deste projeto permitiu consolidar os conhecimentos sobre processos ETL. Através da plataforma KNIME, foi desenvolvido um fluxo completo que integrou dados de clientes, produtos e vendas provenientes de diferentes formatos e fontes, incluindo ficheiros CSV e XML, bem como uma API em formato JSON.

Durante o desenvolvimento, foram aplicadas diversas técnicas de normalização e validação de dados, como por exemplo o uso de expressões regulares, manipulação de texto e controlo de tipos de dados. As junções e cálculos realizados possibilitaram a criação de métricas como o total gasto por cliente, categoria e produto, fornecendo uma visão global do desempenho da loja.

As exportações em formatos diferentes e a gravação na base de dados PostgreSQL garantiram a persistência e disponibilidade da informação, enquanto os gráficos produzidos facilitam a análise visual dos resultados. A estrutura modular, organizada em componentes ETL, contribuiu para um fluxo mais claro e flexível, permitindo a execução e manutenção independentes de cada fase do processo.

No final, o projeto atingiu os objetivos propostos, cumprindo com o objetivo principal de criar um processo de integração completo, automatizado e funcional

Bibliografia

https://fakestoreapi.com/products