Dataset Problem

Question 1

> How would you normalise these data to simplify its processing into a DB ?

Introduction

Keywords:

- Normalize → transform
- Simplify processing → pre-processing
- Database → target

Dataset

> List of Products' prices over 3 years

Products + prices over time

Product attributes

Aa Name	Data Type	Data Group	Data Category	Data Family	Classification
Name	String	Structured	Qualitative		Identifier
Id	Integer	Structured	Qualitative		Identifier
Country	String	Structured	Qualitative		Geographic
Region	String	Structured	Qualitative		Geographic
Variety	String	Structured	Qualitative		Category
Grade	String	Structured	Qualitative		Quality

Aa Name	Data Type	Data Group	Data Category	Data Family	Classification
Price	Float	Structured	Quantitative	Continuous	Economic

Data Schema

- Countries
  - id
  - name
  - continent
- Region
  - id
  - name
  - country_id
- ProductVarieties
  - id
  - name
  - product_id
- ProductGrades
  - id
  - name
  - product_id
- ProductCategories
  - id
  - name
  - index
- Products
  - Id

- name
- variety_id
- grade_id
- product_category_id
- region_id

▼ Prices

- id
- date
- product_id
- min
- max
- avg

Pre processing Process

0. Data discovery

Product attribute

> {ProductName}·(#{NumericIdentifier})

Country attribute

> {CountryName}

Variety

> {ProductName}·/·{”Variety”}·/·{VarietyName}

Grades

> {ProductName}·/·{"Grade"}·/·{GradeName}
> [\n{ProductName}·/·{"Grade"}·/·{GradeName}]

Region

> {CityName}[,·{RegionName}]

Price/Date

> [{PriceValue}]

Date are 7 days apart from each other, therefore, these are `weekly prices over 3 years`

1. Data cleansing

A. Missing Data

- If an entry misses a CountryName and there is a RegionName attribute, query CountryName from the **region attribute** (if possible)
- If an entry misses a RegionName and there is a CountryName, apply "**National**" default region
- If an entry misses a **variety**, apply "common" `← not sure about this one`
- If an entry misses a **grade**, apply "null"

B. Remove noise

- Detect and ignore outlier prices
- If Variety.ProductName is different from the ProductName**,** set **variety** to "common" `← not sure about this one`
- If Grade.ProductName different from the ProductName, set **grade** to "null"

C. Validate data type

- Set to "null" data failing data type check

D. Discarding

- Ignore entries without ProductName
- Ignore entries without both CountryName and RegionName
- Ignore entries without any prices

Data Transformation

A. Formatting

- Clean string values
  - Convert to utf8
  - Remove capitalisation
  - Remove extra spaces
  - Discard “no” in “no. {Number}”
  - replace arabic number by their text version (1 → one) or the opposite, but all must be consistent (example: us n.o. 1 and use one)

    https://www.cde.state.co.us/nutrition/osnffvpproduceinfosheetsapples#:~:text=U.S

- Round numeric value
  - price → 2f rounding

B. Attribute construction

- Exctract variety field → only keep VarietyName
- Extract grade field → only keep GradeName

C. Generalisation

- If region attribute has CityName, check if both match (against geographic API)
  - if it does, discard CityName from **region attribute** value
  - if it doesn’t, get RegionName from geographic API and replace **region attribute** value by result

- Check if RegionName match CountryName (against geographic API)
  - if it does, discard **country attribute**
  - if it doesn't, replace **region attribute** value by **"National"**

D. Aggregation

- Group entries by:
  - product name
  - variety
  - grade
  - country

> Product → Variety → Grade → Country

E. Discretisation

- Create and compute range for group's weekly price:
  - min-price
  - max-price
  - average-price

📌 One product group (i.e `Avocado + Hass + First Quality + Chile`)

📌 Prices date interval are weekly

Question 2

> What additional value can you extract from this dataset ? If you find any please explain how would you collect it (pseudo-algorithm)

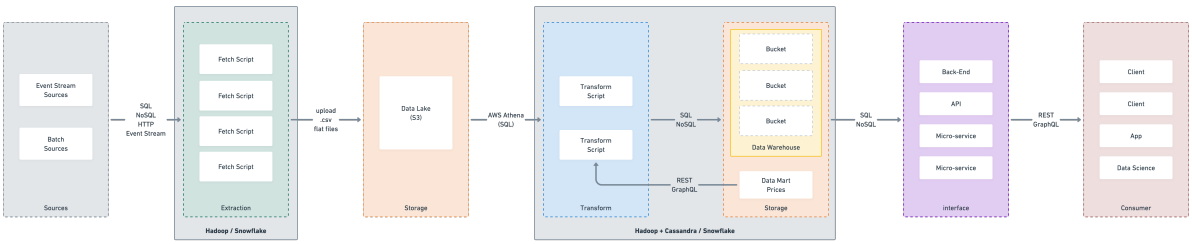
Price

Currency

- If we have the country attr and value is US → apply USD currency
- else if we have available prices in database for this product group (product+variety+grade)
  - compare average/min/max prices in DB to entry price for every currency
    - apply currency with the ratio closest to 1
- else if product+variety+grade has an entry from the US and we have access to countries currencies for given entry (market /economic API)
  - compare US entry price with entry price ratio
    - ratio equal / close to exchange rate ratio → assign expected currency to entry
    - ratio > 0.9 → assign USD as currency

Question 3

> How would you approach the script of putting this information into a database ?(Concurrency, Scale, Prerequisites, etc..)



ELT

Tridge dataset test architecture

https://whimsical.com/tridge-dataset-test-architecture-KA15TqcjP4dbnpfAQMaDHM

📌 The script described above is refered to in the schema as “Transform Script”

Extraction

To allow concurrent / parallel run of the fetch script on several instances, on-premises distributed computing framework like Hadoop or cloud service like Snowflake can be used.

Running extraction scripts on different instances allow faster / immediate fetching (time-sentitive data extraction, like event-streaming or limited-time availability batch data) without risking delay or bandwidth clogging due to excessive load.

Loading

As a temporary / archiving storage place, a Data Lake could be used. That would allow to run our transformation and loeading script at a defined time without risk of missing out, as it could be useful for data scientist that would desire to access raw data, or as archive in case later data audit discover issues in the transform scripts.

AWS S3 offer a out of the box automatic / on-demand scaling logic to increase storing space and data intake capabilities (avoiding slow down or unavailability of the Data Lake)

Transformation (and second loading)

At a desired time (weekly, on a defined day and time), transformation script could be run on different instances — via on-premises distributed computing framework like Hadoop or cloud service like Snowflake — to process data without worrying about slow-down (in case of heavy load due to suddenly high data availability — if any Change Data Capture system is in place)

Data, once transform, would be inserted in a Data Warehouse for later used, interface with service like Apache Cassandra, offering scalability during heavy data

flow and distributed capacities to make the system resilient to bucket/node/shard going offline or encountering bandwidth issue/limitation.

Script

Concerning the script/part of script responsible for directly insert normalized data into the database, script could be assigned different target database / bucket (if not handled by the target itself) via an environment variable containing a node/instance/shard name or url. This would make the script agnostic to the instance it is run on and makes it flexible / generic — for indiscriminate data loading.

In the case of logic-specific loading destination (i.e similar to Kafka Bucket→Topic targeting via EventStream's key), typed/categorized destination's url could be fetched via a dedicated destination DB and would be fetched at the instance start-up to ensured that an up-to-date target-name / urls list is provided to a script.