Tutorial 2

Feature Engineering

Outline

- Setting: ML over relational databases
- Manual feature engineering in relational databases
- Feature Engineering In Practice: SQL + Pandas
- ACORA algorithm

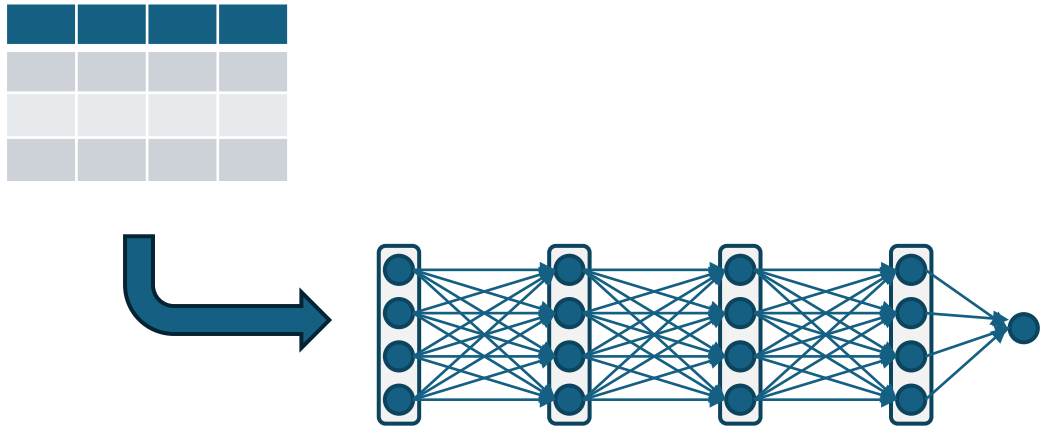
Outline

- Setting: ML over relational databases
- Manual feature engineering in relational databases
- Feature Engineering In Practice: SQL + Pandas
- ACORA algorithm

Tabular Machine Learning

Previously:

- Deep learning network input: a matrix
- Table ⇒ Matrix: Encoding (i.e - 1-hot)

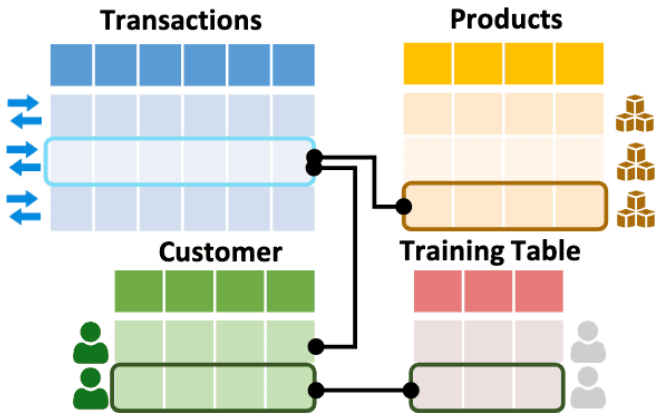


🤔 *Problem*: In reality – more than 1 table

⇒ Database: a collection of tables

⇒ We want to use all the data

💡 *Solution*: Use all the tables to create one table



http://kumo.ai/research/relational-deep-learning-rdl/

Table Definition

Reminder:

- A **tabular signature** (A, dom):
  - $A = (A_1, A_2, \dots, A_n)$ is a set of **attributes**
  - $dom = dom_1 \times \cdots \times dom_n$ is the **domain**
- A **table** (relation) **over** a signature (A, dom) is a subset $r \subseteq dom$

Example:

- $A = (\text{id}, \text{name}, \text{age})$
- $dom = (\mathbb{N}, \text{String}, \mathbb{R})$
- $r = \{(1, \text{Alice}, 17.5), (2, \text{Bob}, 18.5)\}$

$\Rightarrow$

tuple

Visually:

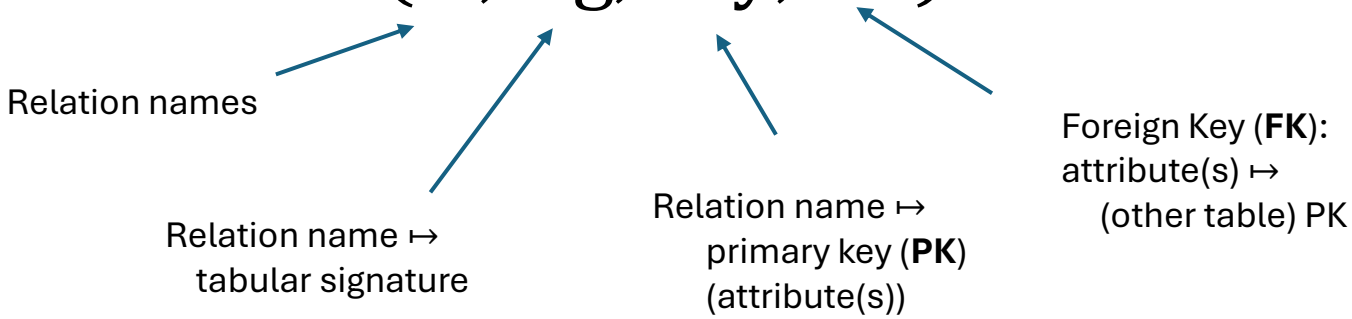
id	name	age
1	Alice	17.5
2	Bob	18.5

tuple/row

Database Definition

Reminder: A **relational schema**

$$\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$$



A relational **database** D over $\mathcal{S}$:

- A mapping from each relation name $R \in \mathcal{R}$ to a table (noted $D(R)$) over $\text{sig}(R)$
  - ✓With PK and FK constraints hold

Database

Example:

Purchases

purchase id	user id (FK)	item id (FK)	purchase date	quantity
100	1	10	2024-01-01	1
101	1	10	2024-01-10	2
102	2	20	2024-01-05	1

Users

user_id	age	country	signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Items

item id	category	price
10	Books	50
20	Electronics	500

Database

Example:

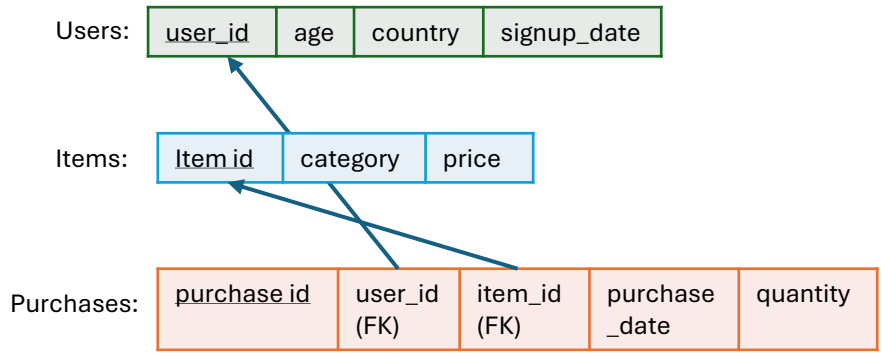
$\mathcal{R} = \{\text{Users}, \text{Items}, \text{Purchases}\}$

$\text{sig}(\text{Users}) = \big((\text{user\_id}, \text{age}, \text{country}, \text{signup\_date}),\ \mathbb{Z} \times \mathbb{Z} \times \text{String} \times \text{Date}\big)$
$\text{sig}(\text{Items}) = \big((\text{item\_id}, \text{category}, \text{price}),\ \mathbb{Z} \times \text{String} \times \mathbb{R}\big)$
$\text{sig}(\text{Purchases}) = \big((\text{purchase\_id}, \text{user\_id}, \text{item\_id}, \text{date}, \text{qty}),\ \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \text{Date} \times \mathbb{Z}\big)$

$\mathcal{S} = (\mathcal{R}, \text{sig}, \text{key}, \mathbf{FK})$

$\text{key}(\text{Users}) = (\text{user\_id})$
$\text{key}(\text{Items}) = (\text{item\_id})$
$\text{key}(\text{Purchases}) = (\text{purchase\_id})$

$\text{Purchases.user\_id} \mapsto \text{Users.user\_id}$
$\text{Purchases.item\_id} \mapsto \text{Items.item\_id}$



Database

Example: Database

$$\mathcal{R} = \{\text{Users}, \text{Items}, \text{Purchases}\}$$

$\{ (1, 25, \text{"Israel"}, \text{"2023} - 10 - 01\text{"}), (2, 40, \text{"USA"}, \text{"2023} - 09 - 15\text{"}) \}$

$\{ (10, \text{"Books"}, 50), (20, \text{"Electronics"}, 500) \}$

$\{ (100, 1, 10, \text{"2024} - 01 - 01\text{"}, 1), (101, 1, 10, \text{"2024}-01-10\text{"}, 2), (102, 2, 20, \text{"2024} - 01 - 05\text{"}, 1) \}$

unique

Purchases

purchase id	user id (FK)	item id (FK)	purchase date	quantity
100	1	10	2024-01-01	1
101	1	10	2024-01-10	2
102	2	20	2024-01-05	1

Users

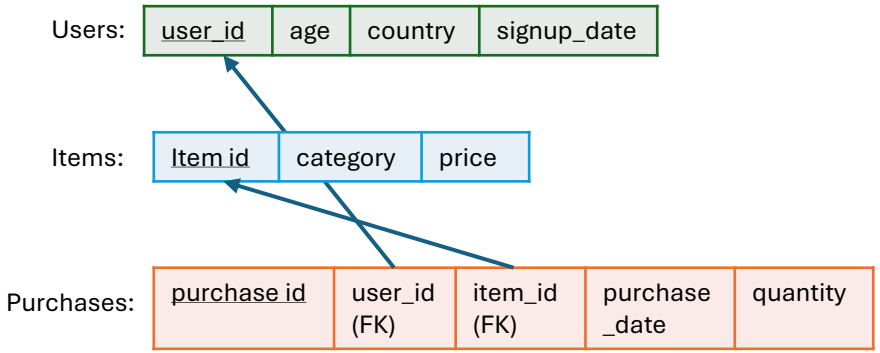
user_id	age	country	signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Items

item id	category	price
10	Books	50
20	Electronics	500

Task Definition

Schema



Task

predict whether a user will make a purchase in the next 30 days

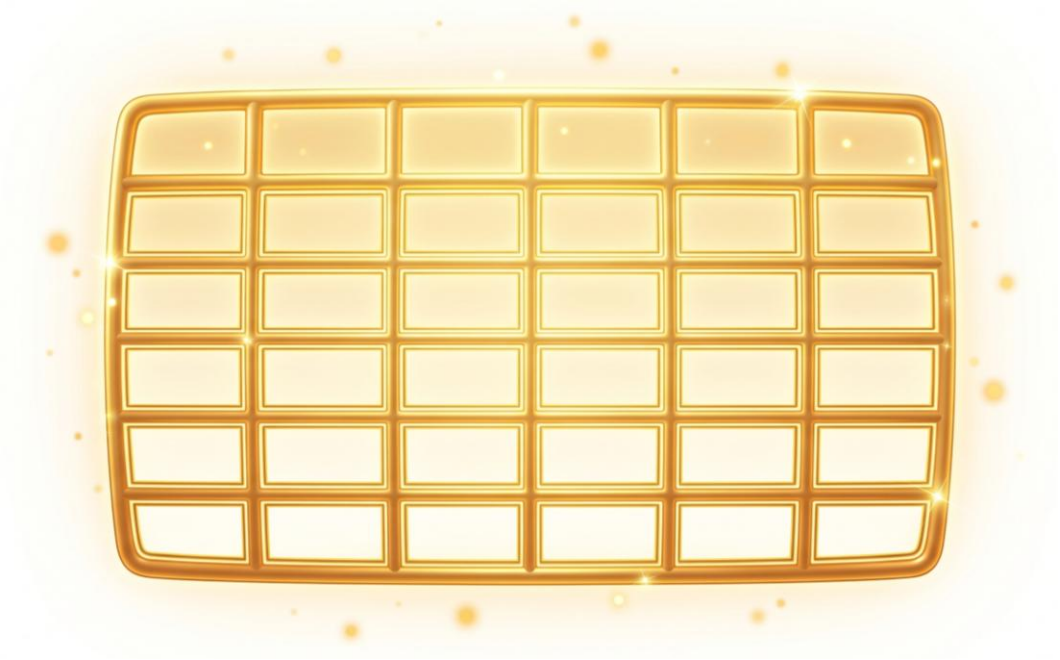
user_id	age	country	signup_date	**Will make a purchase in the next 30 days?**

Outline

- Setting: ML over relational databases
- Manual feature engineering in relational databases
- Feature Engineering In Practice: SQL + Pandas
- ACORA algorithm

Tabular Machine Learning

So let's build a big table!



But… how?

Naive Table Construction

💡 *Idea*: Unite all tables into one

Users

user_id	age	country	signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Items

item id	category	price
10	Books	50
20	Electronics	500

Purchases

purchase id	user id (FK)	item id (FK)	purchase date	quantity
100	1	10	2024-01-01	1
101	1	10	2024-01-10	2
102	2	20	2024-01-05	1

No value

user_id	age	country	signup date	item id	category	price	purchase id	P.user id	P.item id	purchase date	quantity
1	25	Israel	2023-10-01	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	40	USA	2023-09-15	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	10	Books	50	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	20	Electronics	500	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	100	1	10	2024-01-01	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	101	1	10	2024-01-10	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	102	2	20	2024-01-05	1

❌ Sparse and inefficient

❌ Doesn't capture relations between tuples

⇒ information loss

⇒ wastes patterns learning capacity

Feature Engineering

The Core Idea

- Start from the base target table
- Extend it with new valuable information
  - From either the target table or other tables
  - Use queries over the database

Example:

user_id	total spent	days registered
1	5,000$	5
2	10,000$	500

$$\text{spend-per-day} = \frac{\text{total spent}}{\text{days registered}}$$

user_id	total spent	days registered	spend-per-day
1	5,000$	5	1,000
2	10,000$	500	20

✅ Efficiency:
→ Directly encodes complex pattern (i.e tendency to spend)
✅ Generalization:
→ Makes it mathematically easier for the model to learn

Feature Engineering

Feature types

- **Aggregation** – summarize multiple related records into a single value
  - number of purchases
  - total spent
  - minimal purchases
- **Temporal** – capture time related behavior
  - days since last purchase
- **Ratios** – Combine multiple attributes to express a relationship between variables
  - average spent per purchase = num of purchases (aggregation) / account age days (temporal)

Division can't be calculated by the network directly ⇒ must be approximated ⇒ wasteful

Outline

- Setting: ML over relational databases
- Manual feature engineering in relational databases
- Feature Engineering In Practice: SQL + Pandas
- ACORA algorithm

Implementing Feature Engineering

*Two approaches for **how** to do feature engineering in practice*

Declarative Querying

Query language for managing relational databases

- Input: tables directly from DBMS (e.g MySQL)
- Pro: Resides in natural environment
- Best for: Heavy lifting

Imperative Programming

Python library for data manipulation

- Input: DataFrames files of each table
- Pro: Complex data manipulation
- Best for: ML models seamless integration

Constructing Features

Required feature: **total number of past purchases per user**

💡 *Feature type*: Aggregation

```sql
SELECT user_id, COUNT(*) AS num_purchases
FROM Purchases
GROUP BY user_id;
```

Purchases

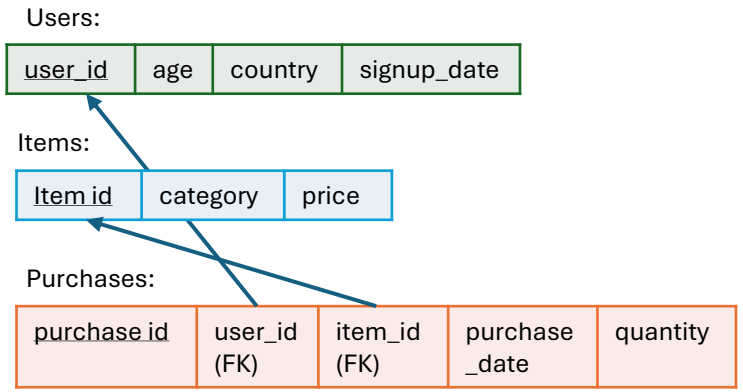
purchase_id	user_id	item_id	purchase_date	quantity
98761	1	5042	2026-04-01	3
98762	2	5043	2025-08-31	1
98763	1	333	2025-04-02	17
98764	1	1501	2025-03-22	5
98765	2	5042	2023-07-11	12

user_id	num_purchases
1	**3**
2	**2**

```python
import pandas as pd

users = pd.read_csv("users.csv")
purchases = pd.read_csv("purchases.csv",
        parse_dates=["purchase_date"])
items = pd.read_csv("items.csv")
```

```python
num_purchases =
    purchases
    .groupby("user_id")
    .size()
    .reset_index(name="num_purchases")
```



Constructing Features

Required feature: **average purchase per user**

💡 *Feature type*: Aggregation

```sql
SELECT
  P.user_id,
  AVG(P.quantity * I.price) AS avg_purchase_value
FROM Purchases AS P
JOIN Items AS I ON P.item_id = I.item_id
GROUP BY P.user_id;
```

Purchases

purchase_id	user_id	item_id	purchase_date	quantity	Category	price
98761	1	5042	2026-04-01	3	Furniture	45.00
98762	2	5043	2025-08-31	1	Accessories	30.25
98763	1	333	2025-04-02	17	Electronics	25.99
98764	1	1501	2025-03-22	5	Electronics	89.50
98765	2	5042	2023-07-11	12	Furniture	45.00

Items

item_id	Category	price
333	Electronics	25.99
1501	Electronics	89.50
5042	Furniture	45.00
5043	Accessories	30.25

```python
purchases_with_price = purchases.merge(items,
                                       on="item_id")
avg_purchase_value =
   purchases_with_price
   .assign(total_price=lambda df:
                                df["quantity"] * df["price"])
   .groupby("user_id")["total_price"]
   .mean()
   .reset_index(name="avg_purchase_value")
```

pandas

user_id	avg_purchase_value
1	**341.443**
2	**285.125**

Constructing Features

Required feature: **days since last purchase**

💡 *Feature type*: Temporal

```sql
SELECT
    U.user_id,
    DATE_PART('day', CURRENT_DATE - MAX(P.purchase_date))
    AS days_since
FROM Users AS U
LEFT JOIN Purchases AS P ON U.user_id = P.user_id
GROUP BY U.user_id;
```


```python
last_purchase = (
    purchases
    .groupby("user_id")["purchase_date"]
    .max()
    .reset_index(name="last_purchase_date")
)

delta_table = (
    users[["user_id"]]
    .merge(last_purchase, on="user_id", how="left")
)

delta_table["days_since"] = (
    pd.Timestamp.today().normalize()
    - delta_table["last_purchase_date"]
).dt.days

delta_table = delta_table[["user_id", "days_since"]]
```

pandas

Purchases

purchase_id	user_id	item_id	purchase_date	quantity
98761	1	5042	2026-04-01	3
98762	2	5043	2025-08-31	1
98763	1	333	2025-04-02	17
98764	1	1501	2025-03-22	5
98765	2	5042	2023-07-11	12

Users

user_id	age	country	signup_date
1	25	Israel	2023-01-15
2	34	USA	2023-02-20
3	28	UK	2023-03-05

user_id	days_since
1	**5**
2	**218**
3	**NULL**

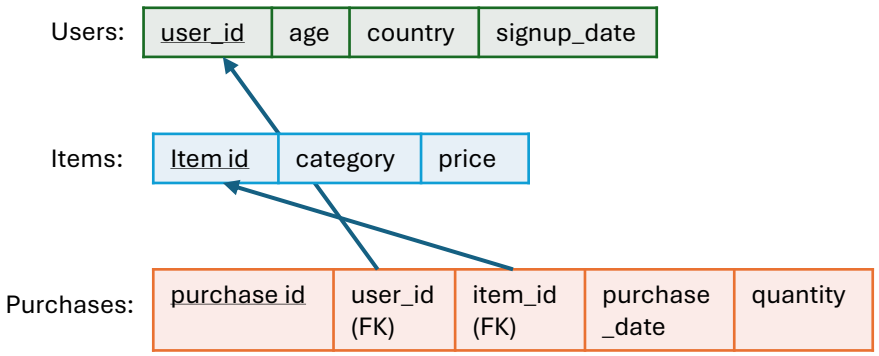
Automated Features Engineering: Query Template

🤔 *Problem*: for every single extracted feature we need to implement a query

💡 *Solution*: Automate feature extraction

→ Instead of designing a concrete query for each feature – design a **query template**

Example:




```
SELECT
   user_id,
   {AGG}(quantity) AS {AGG}_quantity,
   {AGG}(age) AS {AGG}_age
FROM Users
NATURAL JOIN Purchases
GROUP BY user_id;
```

List of aggregated fields

$\mathbf{AGG} \in \{ \text{SUM}, \text{AVG}, \text{MAX}, \text{MIN}, \text{COUNT} \}$

🔨 More ideas of how to automate using templates?

Outline

- Setting: ML over relational databases
- Manual feature engineering in relational databases
- Feature Engineering In Practice: SQL + Pandas
- ACORA algorithm

The ACORA Algorithm

Problem: Still – we need to construct the query templates

👎 Manual

🔦 Can we completely automate the feature engineering process?

ACORA

(Automated Construction of Relational Attributes)

automatically creates useful features from relational databases by combining related tables.

ACORA Algorithm

Input: The domain specification (tables, attributes, types, and an equality relation over types), and a database RDB including a target table T with labeled training objects t and unlabeled test cases.

1. Read specification and build domain graph $\mathcal{G}$
2. Initialize breadth-first list $\mathcal{L}$ with target table: $\mathcal{L} = \{T\}$
3. Initialize feature table F =non-identifier attributes(T)
4. Loop
5. Q = First($\mathcal{L}$)
6. Foreach table R in RDB related to Q in $\mathcal{G}$ through some identifiers ($Q.l$, $R.k$)
7. J = Join R and Q under the condition $R.k$=$Q.l$
8. Foreach attribute $R.j, j \neq k$ (Section 3.3)
9. Foreach target observation t
10. Foreach applicable aggregation operator $\mathcal{A}$
11. Construct $\mathcal{A}(\mathcal{B}_{R.j}(t))$ where $\mathcal{B}_{R.j}(t)$ is the bag of values of attribute $R.j$ related to target entity t.
12. End Foreach
13. Append aggregates $\mathcal{A}$ as attributes to F
14. End Foreach
15. Append J to queue $\mathcal{L}$
16. End Foreach
17. if (stopping criterion) GOTO Select Features
18. End Foreach
19. End Loop
20. Select Features SF from F
21. Build propositional model from SF

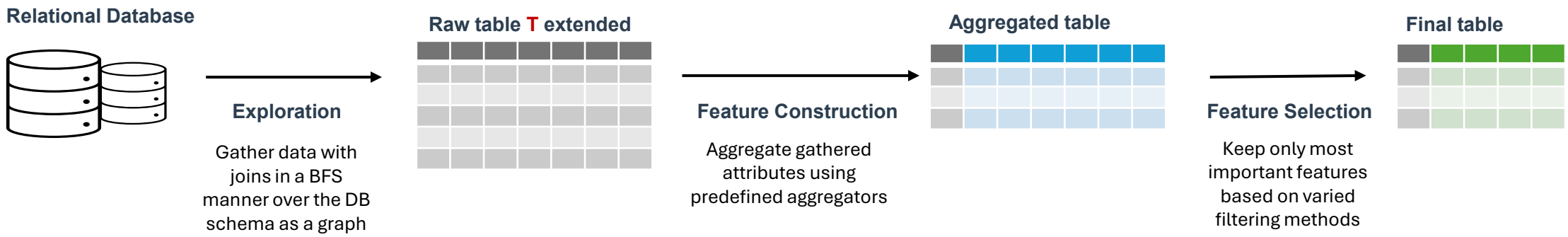
The ACORA Algorithm - Example

Let's use ACORA to automate the feature engineering for the same task as before – predicting whether a user will make a purchase in the next 30 days.

The ACORA Pipeline

Input: Target relation **T**

- For a specific learning task



The ACORA algorithm - Example

Users:

user_id	age	country	Signup date
1	25	Israel	2023-10-01
2	40	USA	2023-09-15

Items:

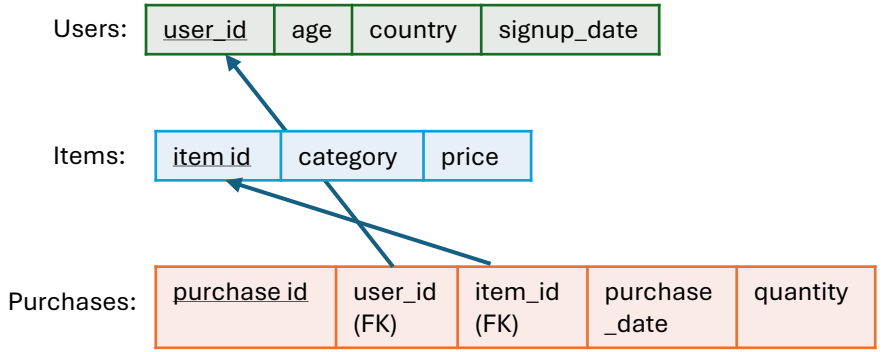
Item id	category	price
10	Books	50
20	Electronics	500

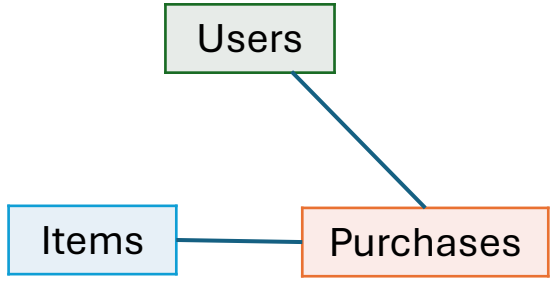
Purchases:

purchase id	User id (FK)	Item id (FK)	Purchase date	quantity
100	1	10	2024-01-01	1
101	1	10	2024-01-10	2
102	2	20	2024-01-05	1

Phase 1 - Exploration

- Input: **Users** relation
  1. Generate the DB schema graph:





- nodes: relation names
- edges: foreign keys

  2. Join according to BFS in the graph:

1. Users ⋈ Purchases (on user id)
2. Users ⋈ Purchases ⋈ Items (on item id)

Natural join notation

💡 *Note*: the graph can be bigger and have cycles. A search count limit should be applied

Phase 1 - Exploration

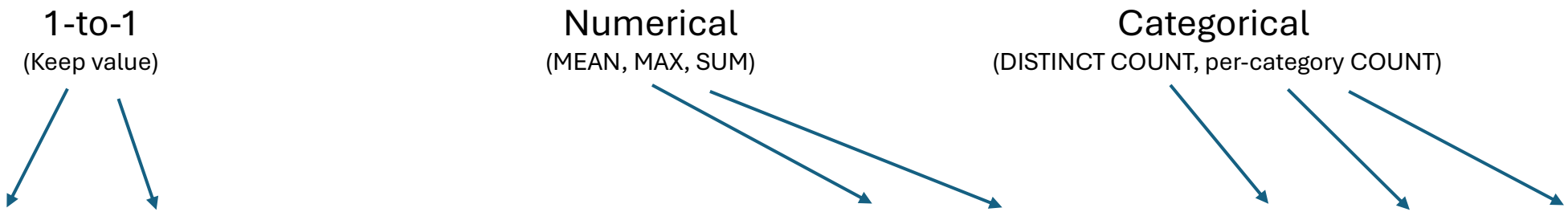
Exploration phase outcome:

user id	age	country	signup date	purchase id	purchase date	item id	category	price	quantity
1	25	Israel	2023-10-01	100	2024-01-01	10	Books	50	1
1	25	Israel	2023-10-01	101	2024-01-10	10	Books	50	2
2	40	USA	2023-09-15	102	2024-01-05	20	Electronics	500	1

Phase 2 – Feature Construction

Now, we group the target relation by its PK and aggregate the feature

- According to predefined aggregators – based on columns metadata, i.e:



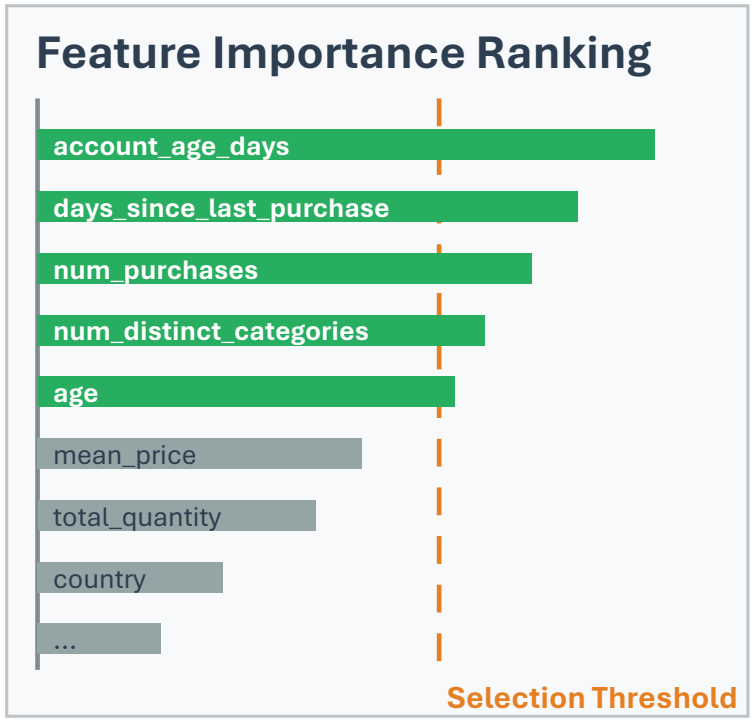
user_id	age	country	account_ age_days	num_ purchases	days_since_ last_ purchase	total_ quantity	mean_ price	max_price	sum_price	num_distinct_cat egories	count_ books	count_ electronics
1	25	Israel	934	2	833	3	50	50	100	1	2	0
2	40	USA	949	1	838	1	500	500	500	1	0	1

Phase 3 – Feature selection

Next, a heuristic or a simple predictive model is used to select the most informative subset of features

Raw Features (12)

1. age
2. country
3. account_age_days
4. num_purchases
5. days_since_last_purchase
6. total_quantity
7. mean_price
8. max_price
9. sum_price
10. count_books
11. count_electronics
12. num_distinct_categories



Selected Features (5)

1. account_age_days
2. days_since_last_purchase
3. num_purchases
4. num_distinct_categories
5. age

💡 *What methods can be applied for feature importance?*

Phase 3 – Feature selection

Feature selection phase outcome:

user_id	age	country	account_ age_days	num_ purchases	days_since_ last_ purchase	total_ quantity	mean_ price	max_price	sum_price	num_distinct_cat egories	count_ books	count_ electronics
1	25	Israel	934	2	833	3	50	50	100	1	2	0
2	40	USA	949	1	838	1	500	500	500	1	0	1

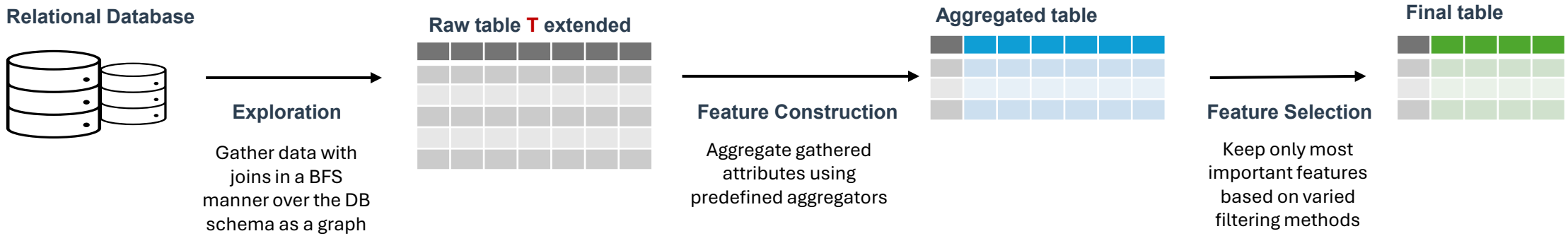
Final Table:

user_id	age	account_ age_days	num_ purchases	days_since_ last_ purchase	num_distinct _categories
1	25	934	2	833	1
2	40	949	1	838	1

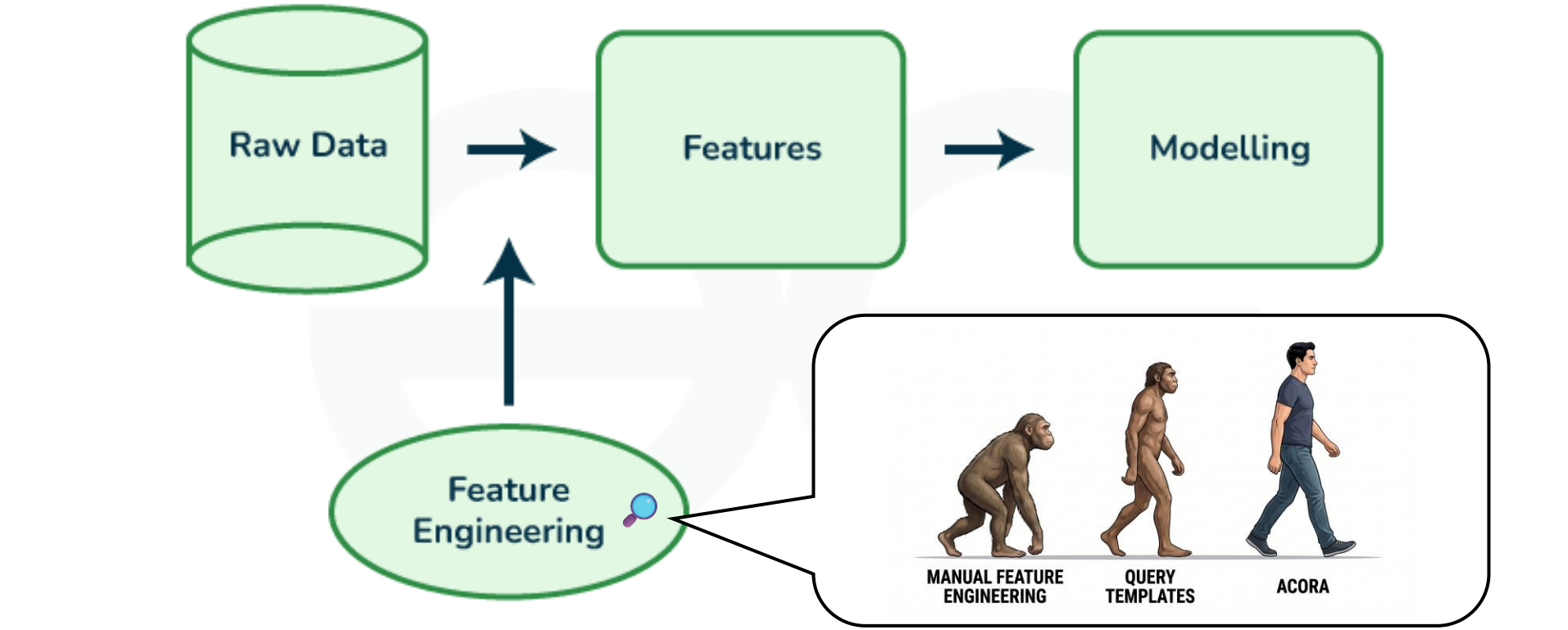
The ACORA Pipeline

Input: Target relation **T**

- For a specific learning task



Summary



https://www.geeksforgeeks.org/machine-learning/advanced-feature-engineering-with-pandas/