

Data Engineering Notes

ibrahim.nasser@fau.de

June 23, 2025

Disclaimer

These are my personal notes and are not official course documents. They may contain inaccuracies or omissions, hence, they should not be considered as a substitute for official course materials or as comprehensive preparation for examinations.

Contents

1	Introduction and Basic Data Types	2
2	Conceptual Modeling	5
3	The Relational Data Model	8
4	Functional Dependencies	10
5	Relational Algebra and SQL	12
6	Normal Forms	17
7	Graph Databases and Cypher Queries	22
8	Descriptive Statistics and Data Normalization	25
9	Distance and Similarity Measures	35

1 Introduction and Basic Data Types

We call the data **Tabular** when there are no modelled dependencies between attributes, for example, demographic attributes such as age, gender, ZIP code, etc. (also called *Nondependency-Oriented Data*). Otherwise it is **Non-Tabular**, e.g. social networks, time series, etc.

Matrix Representation of Data

A set $X = \{X_i \mid i \in \{1 \dots n\}\}$, with n records (samples) is a d -dimensional dataset iff each sample X_i is a set of $\{x_j \mid j \in \{1 \dots d\}\}$ attributes (features). X is tabular if it is invariant w.r.t shuffling of samples and features. Each feature x_j has its own domain \mathcal{D}_j

Quantitative vs. Categorical: A variable x is quantitative (numeric) if its domain \mathcal{D}_x is numeric. Otherwise, Categorical. *Examples (Q):* age, weight, height, BMI, Date of Birth. *Examples (C):* name, gender, country, ZIP Code, weather, ID, day.

Nominal vs. Ordinal: A categorical variable x is ordinal if its domain \mathcal{D}_x has a natural ordering. Otherwise, Nominal. *Examples (N):* weather, name, gender, country, ZIP, ID, day *Examples (O):* heat level, textual gpa.

Finite vs. Infinite: A variable x has a finite domain iff $|\mathcal{D}_x| = N, N \in \mathbb{N}$. Otherwise, Infinite. *Examples (F):* age (years), country, ZIP, ID, gender, day. *Examples (I):* BMI, height, Date of Birth.

Note

All categorical variables have finite domains, not the other way around.

Discrete vs. Cont.: A Quantitative variable x is continuous iff $\forall z, y \in \mathcal{D}_x \exists w \in \mathcal{D}_x, z < w < y$. Otherwise, Discrete. *Examples (D):* age (years, months, days, hours, etc). *Examples (C):* age (unitless, number), Date of Birth (point in cont. time), BMI.

Note

By **rounding** quantitative data, we can transform cont. domains into discrete ones.

Note

Age is quantitative finite discrete if it is computed as whole years, months, days, hours. However, it is quantitative infinite continuous if it is computed as precise value including fractions

Note

Date of Birth is quantitative infinite continuous since it is a point in a continuous endless time

Binary: We call a variable x binary iff $|\mathcal{D}_x| = 2$

Temporal: We call a variable x temporal iff \mathcal{D}_x represents time points or intervals. *Examples:* day, month, Date of Birth

Encoding: Data Encoding refers to the technique of converting data into a form that allows it to be properly used by different systems.

Binning: Binning is an encoding technique that is a function $f : \mathcal{D} \rightarrow \{1 \dots K\}$

Example: Equal-Width Binning: Size (width) of each bin is calculated as $W = \frac{\text{Max}(x) - \text{Min}(x)}{K}$ where K is the number of bins.

One-Hot Encoding

To mitigate the problem of label encoding for nominal variables.

How? Create a fixed-size vector with size = $|\text{unique}(x)|$, where each position corresponds to a unique category value. Assign a 1 to the position representing the category and 0s elsewhere.

Example: Suppose $\text{unique}(x) = \{\text{Red}, \text{Green}, \text{Blue}\}$

- Red $\rightarrow [1, 0, 0]$
- Green $\rightarrow [0, 1, 0]$
- Blue $\rightarrow [0, 0, 1]$

Note

One-hot encoding avoids the problem of implying ordinal relationships. However, it increases dimensionality significantly, especially when the number of categories is large (curse of dimensionality).

Cyclic Encoding

Some categorical variables are *ordinal* and have a natural *cyclic* structure. A classic example is the months of the year:

$$\mathcal{D}_x = \{\text{Jan}, \text{Feb}, \dots, \text{Dec}\}$$

This variable has both an order ($\text{Jan} < \text{Feb} < \dots < \text{Dec}$) and a cyclic relationship (Dec is followed by Jan).

To encode this properly, we use the index i of each category in the ordered list, where $i = 1, 2, \dots, k$, and k is the total number of categories.

Encoding Function:

$$\text{enc}(c_i) = (x_i, y_i)$$

$$x_i = \cos\left(\frac{2\pi(i-1)}{k}\right), \quad y_i = \sin\left(\frac{2\pi(i-1)}{k}\right)$$

This maps each category to a unique point on the unit circle, preserving both order and cyclicity.

Note

Cyclic encoding is useful when the first and last categories are conceptually adjacent (e.g., December and January). This is not possible with standard label or one-hot encoding.

Optional: Normalize to Unit Square

$$\text{enc}(c_i) = \left(\frac{x_i + 1}{2}, \frac{y_i + 1}{2}\right)$$

This scaled version maps points to the square $[0, 1] \times [0, 1]$, which can be useful when input normalization is required for machine learning models. Note that this transformation alters the original unit circle geometry.

Note

Use raw unit circle encoding when preserving angular distance is important. Use the normalized version when the model expects features in the range $[0, 1]$.

Non-Tabular Data

Such as Spatial data, images, time series, string, graphs.

A set $X = \{x_i \mid i \in \{1 \dots n\}\}$ is a d -dimensional **spatial** dataset with n samples if each sample x_i contains a set of $\{x_j \mid j \in \{1 \dots d\}\}$ features AND each data point x_{ij} is associated with a specific spatial location l .

A spatial location l can be a point $(l_x, l_y) \in \mathbb{R}^2$ (2D spatial data) or $(l_x, l_y, l_z) \in \mathbb{R}^3$ (3D spatial data), etc.

Tokenization (Character-Level)

Tokenization is the process of converting raw text into smaller units called tokens. In character-level tokenization, each unique character from the corpus is treated as a token.

Example: Consider the corpus consisting of a single sentence: "hi ai"

- Unique characters: {h, i, , a}
- Assign token IDs: h:0, i:1, :2, a:3
- Tokenized sentence: "hi ai" \rightarrow [0, 1, 2, 3, 1]

Each character in the sentence is replaced by its corresponding token ID.

Graphs

A graph is a mathematical structure used to model pairwise relations between objects.

- A graph G is defined as $G = (V, E)$, where:
 - V is a set of *vertices* (or *nodes*).
 - $E \subseteq V \times V$ is a set of *edges*.

Types of Graphs:

- **Undirected Graph:** An edge $(u, v) \in E$ implies a bidirectional connection:

$$(u, v) \in E \Rightarrow (v, u) \in E$$

- **Directed Graph (Digraph):** Edges have direction:

$$(u, v) \in E \not\Rightarrow (v, u) \in E$$

Graph Representations

Adjacency Matrix:

A $|V| \times |V|$ matrix A , where:

$$A[u][v] = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

- **Space consumption:** $\mathcal{O}(|V|^2)$
- **Edge access:** $\mathcal{O}(1)$
- **Neighbor iteration:** $\mathcal{O}(|V|)$

Adjacency List:

Each vertex $u \in V$ maintains a list of its neighbors.

- **Space consumption:** $\mathcal{O}(|V| + |E|)$
- **Edge access:** $\mathcal{O}(|V|)$ (worst-case search)
- **Neighbor iteration:** $\mathcal{O}(\deg(u))$, where $\deg(u)$ is the degree of vertex u

Weighted Graphs:

In some graphs, each edge $(u, v) \in E$ is associated with a numerical value called a *weight*, often representing cost, distance, capacity, etc.

- For weighted graphs, the edge set becomes:

$$E \subseteq V \times V \times \mathbb{R}$$

or we define a weight function:

$$w : E \rightarrow \mathbb{R}$$

- In the adjacency matrix, $A[u][v]$ stores the weight instead of a binary 0 or 1.
- In the adjacency list, each neighbor can be stored along with its edge weight as a tuple: $(v, w(u, v))$.

2 Conceptual Modeling

ER Model: Entity-Relationship Model is a high-level, conceptual framework to describe entities, their attributes, and the relationships between them.

Entity: Basic concept of the Entity-Relationship (ER) model. It is an object in the real world. E.g. e1 (some employee).

Attribute: Entities have attributes that are the properties that describe them.

Entity Type: All entities that have the same entity type share the same attributes. E.g. EMPLOYEE (type), e1 (Entity).

Attribute Value: A particular entity has a specific value for each of its attributes.

Composite: An attribute is composite if it is described in terms of its smaller parts. E.g. Name, some databases consider name as a composite attribute consisting of two **atomic** attributes First Name and Last Name.

Atomic/Simple: Cannot be divided into smaller parts.

Note

These days, we store **date** as a single value attribute of the type *DATE*. Earlier, date was considered as a composite attribute consisting of atomic attributes *day*, *month*, *year*.

Derived: An attribute is derived if its value is calculated using other **stored** attributes, e.g. age.

Stored: An attribute is stored if it cannot be derived from other attributes.

Note

Age is both derived and atomic. DateOfBirth is both composite and stored.

Single-Valued: An attribute is single-valued if it can have only one value. E.g. DateOfBirth is single-valued composite. Biological sex is single-valued atomic.

Multi-Valued: An attribute is multivalued if it can have several values. E.g. college degrees is multivalued atomic and can have BSc, MSc, BEng, etc.

Note

Affiliation of an entity type RESEARCHER is multivalued (because one can have different affiliations) and composite because an affiliation could be represented as (Org. Name, Dept., Address, Role, Start Date, End Date).

Entity Set: Collection of entities of a particular entity type in a database in a given time point.

Entity Type	Blueprint/Description
Entity Set	Actual set of entities (entity instances) at a point in time

Note

In ancient logic and philosophy we refer to the definition or conceptual content of a term as an *intension*. However, the set of actual things that satisfy a concept is called *extension*. Hence, Entity Type is called intension, Entity Set is called extension.

Candidate Key (Key Attribute): A candidate key is an attribute (or set of attributes) that **uniquely** and **minimally** identifies each entity in an entity set. E.g. StudentID, studentEmail.

Primary Key: A primary key is the chosen candidate key that will be used to uniquely identify entities in the database.

Foreign Key: A foreign key is an attribute in one table/entity that references the primary key of another table/entity. It expresses a relationship between two entity sets.

Composite Primary Key: A composite primary key is a primary key that consists of two or more attributes combined together to uniquely identify a record in a table. Neither attribute alone is sufficient to guarantee uniqueness — but together, they do.

This is common in relationship tables, for example: enrollment relationship between students and courses (M:M):

StudentID	CourseID	Grade
101	CS101	A
101	MATH201	B
102	CS101	B+

Weak Entity Types: Entity types without key attributes.

Strong Entity Types: Entity types with key attributes.

Relations

If we want to model 1:M or M:1 relations, we use the idea of foreign key (modeling the relation with single value attribute). Examples:

PersonID	PersonName	categoryID	categoryName	catID	catName	ownerID	articleID	title	categoryID
1	Alice	1	Sport	1	Daisy	1	1	title	1
2	Bob	2	Science	2	Smart	2	2	title	2
				3	Sweet	1	3	title	1

We model M:M Relations by creating an entity representing that relation (usually with composite primary key).

Relationship Type: The definition / template / blueprint of the relationship

Relationship Instance: A single actual link between entities.

Relationship Set: The collection of all relationship instances at a given time.

Participation: We say that entity types $E_1 \dots E_n$ participate in the **relationship type** R .

Relationship Degree: Number of participating entity types in the relation. E.g. consider a relation SUPPLY that models which suppliers supply which projects and what parts are supplied, the degree here is 3 due the three entity types (SUPPLIER, PROJECT, PART).

Role Name: The name describing the part an entity plays in a relationship.

Recursive Relationship: A relationship where the same entity type participates more than once with different roles. E.g. SUPERVISION.

Cardinality Ratio: Specifies the maximum number of entities of one type that can be associated with an entity of another type in a relationship. Examples (E_1 BINARY_RELATION E_2):

Let E_1 and E_2 be the sets of entities of type E_1 and E_2 , respectively. Let $R \subseteq E_1 \times E_2$ be the binary relation between them: $R = \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2\}$

- **1:1** each entity of type E_1 can be related to at most one entity of type E_2 and vice versa

$$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \leq 1$$

$$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \leq 1$$

- **1:M** one entity of type E_1 can be related to many entities of type E_2 . Each entity of type E_2 can be related to at most one entity of type E_1

$$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \geq 0$$

$$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \leq 1$$

- **M:1** Inverse of **1:M**

$$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \leq 1$$

$$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \geq 0$$

- **M:M** many entities of type E_1 can be related to many entities of type E_2 and vice versa

$$\forall e_1 \in E_1, \quad |\{e_2 \in E_2 \mid (e_1, e_2) \in R\}| \geq 0$$

$$\forall e_2 \in E_2, \quad |\{e_1 \in E_1 \mid (e_1, e_2) \in R\}| \geq 0$$

Total vs. Partial Participation

- **Total Participation of E_1 in R**
 $\forall e_1 \in E_1, \exists e_2 \in E_2 : (e_1, e_2) \in R$ (Every entity of E_1 is related to at least one entity of E_2 .)
- **Partial Participation of E_1 in R**
 $\exists e_1 \in E_1 : \forall e_2 \in E_2, (e_1, e_2) \notin R$ (There exists an entity in E_1 that does not participate in R .)

Migrating attributes from relations to entities:

- 1:1 relationship types: Attributes can be migrated to either participating entity. (e.g. EMP MANAGES DEPT, start_date)
- 1:N or N:1 relationship types: Attributes should be migrated to the entity that participates at most once. (e.g. EMP WORKS_FOR DEPT, start_date)
- M:N relationship types: Attributes cannot be migrated to the participating entities and must remain on the relationship itself.

Identifying Relationships: a special relationship where a weak entity is identified by its relationship with a strong entity. The weak entity cannot exist without the strong entity, and the relationship plays a crucial role in providing the weak entity with a composite key.

Existence Dependency: A weak entity depends on the strong entity for its existence. It cannot exist without being related to a strong entity.

Note

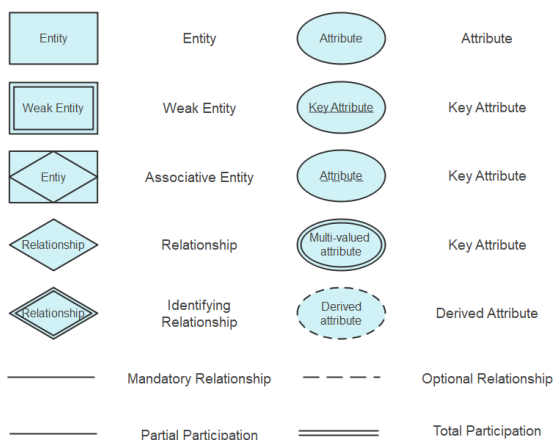
A weak entity type has total participation in its identifying relationship. This means that every instance of the weak entity must be associated with at least one instance of the strong entity. If it doesn't, the weak entity doesn't exist ("existence dependency").

Example:

- Consider we have the following strong entities, customer, product.
- To manage orders, we have two entities, order, orderItem.
- Order is a strong entity since each order has its ID
- However, OrderItem entity can have OrderID, LineNumber, ProductID, quantity, price, discount, etc.
- In this case, OrderItem entity is a weak one, it cannot exist unless an order exists, therefore, the primary key is composite (OrderID, LineNumber)

Min-Max Modeling: Given an entity E participating in Relation R . If at least min and at most max instances of E must participate in R with $min \geq 0, max \geq 1, max \geq min$, then we say E respects min-max constraint (min, max) w.r.t R .

ER Diagram



3 The Relational Data Model

Set: A set is a well-defined collection of distinct objects, considered as an object in its own right. The objects in a set are called elements or members. Sets are usually denoted by capital letters like A , B , or S , and elements are listed within curly braces. For example, $A = \{1, 2, 3\}$ is a set containing the numbers 1, 2, and 3.

Element of a Set: If x is an element of set A , we write $x \in A$. If x is not in A , we write $x \notin A$.

Set-builder Notation: Set-builder notation is a shorthand used to describe a set by stating the properties that its elements must satisfy. For example: $\{x \in \mathbb{N} \mid x \text{ is even}\}$ describes the set of even natural numbers.

Cardinality: The cardinality of a set is the number of elements in the set, denoted $|A|$. For example, if $A = \{1, 2, 3\}$, then $|A| = 3$.

Cartesian Product: Let A and B be sets, then $A \times B = \{(a, b) \mid a \in A, b \in B\}$, e.g. $A = \{1, 2\}, B = \{x, y\}, A \times B = \{(1, x), (1, y), (2, x), (2, y)\}$

Subset: $A \subseteq B \Leftrightarrow \forall X. X \in A \Rightarrow X \in B$

Proper Subset: $A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$

Relation: $R \subseteq A \times B$. If $(a, b) \in R$, we say that a is related to b via R

Left Total Relation: A relation $R \subseteq A \times B$ is left total (total on A) iff each element in A is related to at least one element in B . $\forall a \in A. \exists b \in B. (a, b) \in R$

Right Unique Relation: A relation $R \subseteq A \times B$ is right unique iff each element in A is related to at most one element in B . $\forall a \in A, \forall b_1, b_2 \in B, ((a, b_1) \in R, (a, b_2) \in R) \Rightarrow b_1 = b_2$

Function: A function is left total and right unique relation. $f : A \rightarrow B$

Partial Function: A partial function is a right unique relation, **not** necessarily left total. $f \rightharpoonup B$

Set Union: $x \in A \cup B \Leftrightarrow x \in A \text{ or } x \in B$

Set Intersection: $x \in A \cap B \Leftrightarrow x \in A \text{ and } x \in B$

Disjoint: We say two sets A, B are disjoint iff $A \cap B = \phi$

Relation Schema: A declaration $R(A_1:D_1, \dots, A_n:D_n)$ consisting of a name R , a finite, non-empty attribute set $\{A_i\}$ and, for each attribute, its domain $\text{dom}(A_i) = D_i$.

Schema Satisfaction: A tuple $t = (v_1, \dots, v_n)$ satisfies the schema if $v_i \in D_i \forall i$.

Types: Classes of atomic values that share representation and operations, e.g. **Int**, **Real**, or **String**.

Domain: A set of atomic values with application-specific semantics whose underlying implementation type is fixed. Domains may define default values. Example: $\text{EmployeeAge} = \text{Int}[18, 65]$.

Note

Domain declaration examples: `Name = String(20)`, `DollarPrice = Decimal(5,2)`.

Instance: A finite set of tuples that all satisfy a given relation schema. While the schema is comparatively stable (static), its instance is *dynamic*: it evolves through insertions, deletions, and updates.

Two Equivalent Views on Tuples

- **Positional (Cartesian-product) view:** t is an ordered list (v_1, \dots, v_n) . Column order carries meaning; attribute names are implicit.
- **Functional view:** Fix $A = \{A_1, \dots, A_n\}$ and $D = \bigcup_i D_i$. Then a tuple is a function $t : A \rightarrow D$ with $t(A_i) \in D_i$. Here, order is irrelevant and attribute names are explicit.

Domain Constraint: Each attribute value must lie in its declared domain D_i . Usually enforced by the DBMS type checker.

Functional Dependency (FD): For attribute sets $X, Y \subseteq A$, the notation $X \rightarrow Y$ states: for any two tuples t_1, t_2 , equality of X -values implies equality of Y -values. Written out: $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$.

Superkey: An attribute set K with $K \rightarrow A$ (it functionally determines the whole tuple).

Candidate Key: A minimal superkey — removing any attribute from it destroys the functional determination of A .

Primary Key: The candidate key chosen by the database designer to serve as the principal identifier of tuples in a relation. Remaining candidate keys are called *alternate keys*.

Example

- Relation schema *Employee*(*EmpID*, *SSN*, *Email*, *Name*, *Dept*).
- *Superkeys* include any attribute set that uniquely identifies tuples, e.g. {*EmpID*}, {*SSN*}, {*EmpID*, *Name*}. The third set still determines the whole tuple but is *not* minimal.
- *Candidate keys*: the minimal superkeys {*EmpID*} and {*SSN*}. Each is irreducible.
- *Primary key*: suppose we designate *EmpID* as the primary key. The other candidate becomes an *alternate key* available for unique look-ups.

Foreign Key: Attribute(s) in relation *R* whose values must also appear as the primary-key values of another relation *S* (ensuring referential integrity).

Note

When an insertion, deletion or modification would break any constraint, the DBMS may (i) reject the change or (ii) repair it automatically (“cascade”, insert default/null, etc.). The exact behaviour is part of the schema definition.

A word on modeling different cardinalities

Relational databases use foreign keys (FKs) to represent associations between entities. The modeling depends on the cardinality:

One-to-One (1:1)

A FK is placed in one of the tables.

Person	ID (PK)	Name	PassportID (FK)	Passport	ID (PK)	Number	DateOfIssue
	1	Alice	101		101	X1234	2020-01-01

One-to-Many (1:M) or Many-to-One (M:1)

The FK is placed in the table on the “many” side, referencing the “one” side.

Department	ID (PK)	Name	Employee	ID (PK)	Name	DepartmentID (FK)
	1	Human Resources		101	John	1

Many-to-Many (M:M)

Modeled via a relation table with two FKs, each referencing one of the related tables. The combination of FKs often serves as the primary key.

StdID	StdName	CourseID	CourseTitle	StdID	CourseID
1	Jane	10	Database Systems	1	10
2	Mark	11	Operating Systems	1	12
3	Sara	12	Algorithms	2	10
				2	11
				3	11

4 Functional Dependencies

Prime Attribute: An attribute that is part of any candidate key.

Nonprime Attribute: An attribute that is not part of any candidate key.

Example: consider the simple relation STUDENT(ID, Email, Name, Phone, CourseID). Possible super keys:

$\{ID\}, \{Email\}, \{ID, Name\}, \{ID, Email, Phone\}$

Only $\{ID\}$ and $\{Email\}$ are prime attributes.

Trivial FD: We say that a functional dependency $X \rightarrow Y$ is **trivial** iff $Y \subseteq X$.

Full FD: A functional dependency $X \rightarrow Y$ is full iff for any $A \in X$, $(X - \{A\}) \rightarrow Y$ does not hold, i.e. you cannot remove any attribute from X without breaking the dependency. Example: $\{studentID, CourseID\} \rightarrow Grade$.

Partial FD: A functional dependency $X \rightarrow Y$ is partial iff $\exists A \in X$. $(X - \{A\}) \rightarrow Y$ holds.

Transitive FD: A functional dependency $X \rightarrow Y$ is transitive in a relation R iff \exists **Nonprime set of attributes** $Z \in R$ and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

Inference: A functional dependency $X \rightarrow Y$ is **inferred** from a set of functional dependencies F on a relation R iff $X \rightarrow Y$ holds in every instance of R that satisfies all dependencies in F

Armstrong's Inference Rules for Functional Dependencies

- **IR1:** $(Y \subseteq X) \Rightarrow (X \rightarrow Y)$ (reflexive)
- **IR2:** $(X \rightarrow Y) \Rightarrow (X \cup Z \rightarrow Y \cup Z)$ (augmentation)
- **IR3:** $((X \rightarrow Y) \wedge (Y \rightarrow Z)) \Rightarrow X \rightarrow Z$ (transitive)

Closure: The closure of attribute set X under a set of functional dependencies F , denoted as X_F^+ is the set of all attributes that X can determine using FDs in F . $X_F^+ = \{A \mid X \rightarrow A \in F \text{ or can be inferred from it}\}$

Closure Algorithm

- input: a set F of FDs on a relation R , and a set of attributes X contained in R
- initialization: $X_F^+ = X$
- changed = True
- while changed:
 1. changed = False
 2. for each FD $Y \rightarrow Z \in F$:
 - (a) If $(Y \subseteq X_F^+) \wedge (Z \notin X_F^+)$:
 - i. $X_F^+ = X_F^+ \cup \{Z\}$
 - ii. changed = True
- Output: X_F^+

FDs Verification: F implies $X \rightarrow Y$ iff $Y \subseteq X_F^+$

Superkeys Verification: X is a super key for R with attribute set U iff $X_F^+ = U$

Finding Candidate Keys

Input: Relation (R) over set of attributes (U), set of FDs (F)

initialization: $K := U$

minimal = False

while not minimal

 minimal = True

 for each attribute A in K

 compute closure of (K-A) under F

 if the closure = U

 set $K := K - \{A\}$

 minimal = False

Return: K

Coverage: For any two sets of functional dependencies F_1, F_2 , we say that F_1 covers F_2 , iff $\forall X \rightarrow Y \in F_2. Y \subseteq X_{F_1}^+$.

Equivalence: For any two sets of functional dependencies F_1, F_2 , we say they are equivalent, iff they cover each other.

Example: verify whether the following FDs are equivalent.

- $F_1 = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ and
- $F_2 = \{A \rightarrow CD, E \rightarrow AH\}$

We check first if F_1 covers F_2

- considering $A \rightarrow \{C, D\}$ $\{C, D\} \in A_{F_1}^+ = \{A, C, D\}$
- considering $E \rightarrow \{A, H\}$ $\{A, H\} \in E_{F_1}^+ = \{E, A, D, H, C\}$
- Hence F_1 covers F_2

Next, we check first if F_2 covers F_1

- considering $A \rightarrow C$ $C \in A_{F_2}^+ = \{C, D\}$
- considering $\{A, C\} \rightarrow D$ $D \in \{A, C\}_{F_2}^+ = \{A, C, D\}$
- considering $E \rightarrow \{A, D\}$ $\{A, D\} \in E_{F_2}^+ = \{A, H, C, D\}$
- considering $E \rightarrow H$ $H \in E_{F_2}^+ = \{A, H, C, D\}$
- Hence F_2 covers F_1

Therefore, they are equivalent

Redundancy: A functional dependency $f = X \rightarrow A$ is redundant in FDs set F iff $A \subseteq X_G^+$ where $G = F - \{X \rightarrow A\}$, i.e. $F - \{f\}$ implies f .

Extraneous: Given a set F of FDs and one $f = AX \rightarrow B \in F$, then A is extraneous if $B \subseteq X_F^+$

Minimal cover: A set of FDs F is a minimal cover of a set of FDs E iff F covers E and there is no $f \in F$. $F - \{f\}$ covers E

Canonical: A functional dependency $f = X \rightarrow Y$ is in a canonical form iff $|Y| = 1$

Minimal set of FDs: A set F of FDs is minimal iff it satisfies the following conditions: (i) All FDs in a canonical form. (ii) No extraneous attributes. (iii) No redundant FDs.

Steps to Obtain a Minimal Set of Functional Dependencies

1. Transform to Canonical form
2. Remove Extraneous Attributes
3. Remove Redundant FDs

5 Relational Algebra and SQL

Data Model: In relational databases, the data model specifies how data is structured and how it can be manipulated. i.e. it says that data is organized into tables (called relations) with columns (attributes) and rows (tuples).

Relational model: In relational databases, the relational model represent data as relations (tables). Each relation has constraints, such as keys or data types, to ensure data integrity.

Relational Algebra: The formal system for manipulating relations. It provides a theoretical foundation for **Query** operations used in relational databases

Algebra: A formal system in which expressions are constructed using operators and atomic operands. These expressions can be evaluated, and two expressions are considered equivalent if they yield the same result for all possible values of their operands.

Relational Algebra: A type of algebra where the operands are relations (tables), and the operators are defined for any instance of those relations. Operations can be combined to form complex expressions, and evaluating an expression produces a result schema (the structure of the output) and a result instance (the actual data produced).

SQL: The Standard Query Language (SQL) is the language used to interact with relational databases. It is a **declarative** language, meaning that when you write a query, you describe what result you want, not how the database should compute it. This contrasts with procedural languages, where you must specify every step.

SQL Structure

SQL is organized into several sub-languages, each serving a distinct purpose:

- Data Definition Language (DDL): used to define or alter the structure of database objects. Commands used such as: CREATE, ALTER, DROP
- Data Manipulation Language (DML): used to retrieve and manipulate data. Command used such as: SELECT, UPDATE, INSERT, DELETE
- Data Control Language (DCL): manages user permissions and access control. Commands used such as: GRANT, REVOKE
- Transaction Control Language (TCL): manages database transactions. Commands used such as: COMMIT ROLLBACK

Example DML Queries:

```
SELECT name, age FROM Student WHERE age >= 18 ORDER BY name ASC;

INSERT INTO Student (stdId, name, age) VALUES (101, "Alice", 20);

UPDATE Student SET age = age + 1 WHERE stdId = 101;

DELETE FROM Student WHERE stdId = 101;
```

Examples on COUNT, DISTINCT, EXIST, IN

```
SELECT COUNT(*) FROM Student;

SELECT COUNT(DISTINCT stdID) FROM Student;

SELECT * FROM employees e
WHERE EXISTS (
    SELECT 1
    FROM bonus b
    WHERE b.employee_id = e.employee_id
);

SELECT * FROM employees WHERE department_id IN (1, 2, 5);
```

Example DDL Queries:

```
CREATE TABLE Course (courseID INT PRIMARY KEY,  
                      title    VARCHAR(100)  
                      );
```

```
ALTER TABLE Course ADD COLUMN credits INT;
```

```
DROP TABLE Course;
```

Example DCL Queries:

```
GRANT SELECT, INSERT ON Student TO user1;
```

```
REVOKE INSERT ON Student FROM user1;
```

Example TCL Queries:

```
BEGIN;  
UPDATE Account SET balance = balance - 100 WHERE id = 1;  
COMMIT;
```

Example Primary Key and Foreign Key:

```
CREATE TABLE Department (  
    deptID INT PRIMARY KEY,  
    deptName VARCHAR(100)  
);
```

```
CREATE TABLE Employee (  
    empID INT PRIMARY KEY,  
    empName VARCHAR(100),  
    deptID INT,  
    FOREIGN KEY (deptID) REFERENCES Department(deptID)  
);
```

Example Composite Primary Key:

```
CREATE TABLE Student (  
    stdID INT PRIMARY KEY,  
    stdName VARCHAR(100)  
);
```

```
CREATE TABLE Course (  
    crsID INT PRIMARY KEY,  
    crsName VARCHAR(100)  
);
```

```
CREATE TABLE Enrollment (  
    stdID INT,  
    crsID INT,  
    grade CHAR(2),  
    PRIMARY KEY (stdID, crsID),  
    FOREIGN KEY (stdID) REFERENCES Student(stdID),  
    FOREIGN KEY (crsID) REFERENCES Course(crsID)  
);
```

Example Domain Constraints:

```
CREATE TABLE Product (  
    id INT PRIMARY KEY,  
    price DECIMAL(10,2) CHECK (price >= 0),  
    category VARCHAR(50) NOT NULL  
);
```

Set Operators

Arity / Degree: let $R(A_1, \dots, A_n)$ be a relation schema, the arity of R is $\text{arity}(R) = n$

Union Compatible: We say that relations $R(A_1, \dots, A_n)$ and $S(B_1, \dots, B_n)$ are union compatible if $\text{arity}(R) = \text{arity}(S)$ and $\text{dom}(A_i) = \text{dom}(B_i) \quad \forall i \in \{1, \dots, n\}$

Relation Union: $R_1 \cup R_2 = \{t \mid t \in R_1 \vee t \in R_2\}$. SQL Equiv.: `SELECT * FROM R UNION SELECT * FROM S`

Relation Intersection: $R_1 \cap R_2 = \{t \mid t \in R_1 \wedge t \in R_2\}$. SQL: `SELECT * FROM R INTERSECT SELECT * FROM S`

Relation Difference: $R_1 - R_2 = \{t \mid t \in R_1 \wedge t \notin R_2\}$. SQL: `SELECT * FROM R EXCEPT SELECT * FROM S`

Relation Cartesian Product: $R_1 \times R_2 = \{t_1 \circ t_2 \mid t_1 \in R_1, t_2 \in R_2\}$. SQL: `SELECT * FROM R CROSS JOIN S`

Relation Operators (Unary)

Rename: Changes the schema of the relation R by renaming attribute A_1 to B_1 and so on. $\rho_{(B_1, \dots, B_n \leftarrow A_1, \dots, A_n)}(R)$. SQL: `SELECT a AS b FROM R AS R1`

Selection: $\sigma_C(R) = \{t \in R \mid C(t)\}$ is the set of all tuples in R that satisfy the condition C . The condition C is a Boolean expression composed of predicates:

$$C = P_1 \text{ op}_1 P_2 \text{ op}_2 \dots \text{ op}_{n-1} P_n$$

where each operator $\text{op}_i \in \{\text{AND}, \text{OR}, \text{NOT}\}$, and each predicate P_i has the form:

$$P_i ::= A \theta B \quad \text{or} \quad A \theta c$$

with attributes A, B , constant c , and comparison operator $\theta \in \{=, <, \leq, >, \geq, \neq\}$.

SQL Equiv. : `SELECT * FROM R WHERE C`

Projection: $\Pi_Y(R) = \{t[Y] \mid t \in R\}$. SQL: `SELECT Y FROM R`

Idempotent: An operator \mathcal{O} is called *idempotent* if applying it multiple times has the same effect as applying it once: $\mathcal{O}(\mathcal{O}(x)) = \mathcal{O}(x) \quad \forall x$. **Projection** is *Idempotent*

Relation Operators (Binary)

Theta Join: $R_1 \bowtie_C R_2 = \{t_1 \circ t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2 \wedge C(t_1 \circ t_2)\} = \sigma_C(R_1 \times R_2)$

Example: `SELECT * FROM Employee JOIN Bonus ON Employee.salary > Bonus.threshold`

id	salary	threshold	id	salary	threshold
1	50000	20000	1	50000	20000
2	30000	40000	1	50000	40000
			2	30000	20000

Equi-Join: Theta Join with C consists only of equality comparison.

Example: `SELECT * FROM Orders JOIN Customers ON Orders.cust_id = Customers.id`

order_id	cust_id	id	name	order_id	cust_id	id	name
101	1	1	Alice	101	1	1	Alice
102	2	2	Bob	102	2	2	Bob
103	4	3	Carol				

Natural Join: Equi-Join where C is quality on common attributes and duplicate common attributes are removed from the result.

Example: `SELECT * FROM Employee NATURAL JOIN Department`

emp_id	dept_id	dept_id	name	emp_id	dept_id	name
1	10	10	HR	1	10	HR
2	20	20	IT	2	20	IT
3	10	30	Sales	3	10	HR
4	40	60	Legal			
5	50					

Outer Join: Natural Join but preserves unmatched tuples by padding them with NULL values. That can be done on the relation on the left, right or both.

Left Outer Join: Includes all tuples from the left relation, padding unmatched right-side tuples with NULLS.

Example: `SELECT * FROM Orders LEFT OUTER JOIN Customers ON Orders.cust_id = Customers.id`

order_id	cust_id	id	name	order_id	cust_id	id	name
101	1	1	Alice	101	1	1	Alice
102	2	2	Bob	102	2	2	Bob
103	4	3	Carol	103	4	NULL	NULL

Right Outer Join: Includes all tuples from the right relation, padding unmatched left-side tuples with NULLS.

Example: `SELECT * FROM Orders RIGHT OUTER JOIN Customers ON Orders.cust_id = Customers.id`

order_id	cust_id	id	name	order_id	cust_id	id	name
101	1	1	Alice	101	1	1	Alice
102	2	2	Bob	102	2	2	Bob
103	4	3	Carol	NULL	NULL	3	Carol

Full Outer Join: Includes all tuples from both relations, padding unmatched tuples from either side with NULLS.

Example: `SELECT * FROM Orders FULL OUTER JOIN Customers ON Orders.cust_id = Customers.id`

order_id	cust_id	id	name	order_id	cust_id	id	name
101	1	1	Alice	101	1	1	Alice
102	2	2	Bob	102	2	2	Bob
103	4	3	Carol	103	4	NULL	NULL
				NULL	NULL	3	Carol

Operators Properties

Operator	Result Schema	Result Size	Comm.	Assoc.	Idem.	Duplicates
Union	Same as inputs	$\leq R + S $	Yes	Yes	No	No
Intersection	Same as inputs	$\leq \min(R , S)$	Yes	Yes	Yes	No
Difference	Same as R	$\leq R $	No	No	No	No
Cartesian Product	$R \cup S$	$ R \cdot S $	Yes	Yes	No	Yes
Rename	Same	$ R $	N/A	N/A	N/A	N/A
Selection	Same as R	$\leq R $	Yes	Yes	No	Yes
Projection	Subset of R	$\leq R $	No	No	Yes	No
Theta Join	$R \cup S$	$[0, R \cdot S]$	No	No	No	Yes
Equi-Join	$R \cup S$	$[0, R \cdot S]$	No	No	No	Yes
Natural Join	$R \cup S \setminus C$	$[0, R \cdot S]$	Yes	Yes	No	Yes
Left Outer Join	$R \cup S$	$\geq R $	No	No	No	Yes
Right Outer Join	$R \cup S$	$\geq S $	No	No	No	Yes
Full Outer Join	$R \cup S$	$\geq \max(R , S)$	No	No	No	Yes

Common Datatypes

Category	Common Data Types
Numeric	INT, BIGINT, DECIMAL(p,s), FLOAT, DOUBLE
Character	CHAR(n), VARCHAR(n), TEXT
Date/Time	DATE, TIME, TIMESTAMP, DATETIME
Boolean	BOOLEAN / BOOL

Algebra to SQL Example

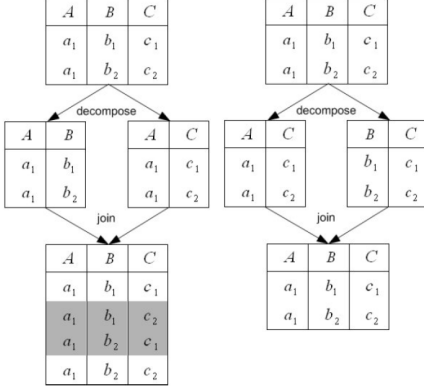
```

$$\pi_{\text{supplier\_name, product\_name}} (\sigma_{\text{price} > 50} ((\text{Suppliers} \bowtie_{\text{supplier\_id}} \text{Products}) \bowtie_{\text{category\_id}} \text{Categories}))$$
  
WITH SupplierProducts AS (  
  SELECT  
    s.supplier_name,  
    p.product_name,  
    p.price,  
    p.category_id  
  FROM Suppliers s  
  JOIN Products p ON s.supplier_id = p.supplier_id  
) ,  
Filtered AS (  
  SELECT * FROM SupplierProducts WHERE price > 50  
)  
SELECT f.supplier_name, f.product_name  
FROM Filtered f  
JOIN Categories c ON f.category_id = c.category_id;
```


6 Normal Forms

Normalization: Decomposing a large schema $R(A)$ into smaller schemata $R_i(A_i)$ with $A_i \subset A$ to minimize redundancy, avoid information loss and preserve functional dependencies.

Spurious Tuples: Decomposing a relation into multiple relations as part of normalization can lead to spurious tuples when these relations are rejoined (tuples that did not exist in the original relation). **Example**



Lossless-Join Decomposition: We call a decomposition of $R(A, B, C)$ into $R_1(A, B)$ and $R_2(B, C)$ lossless-join iff, for all instances r of R that respect the FDs, the following identity holds: $r = \Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

Example (Lossless): Consider the relation: `empDept(empID, name, deptID, deptName)` with FDs:

$\text{empID} \rightarrow \text{name}$

$\text{empID} \rightarrow \text{deptID}$

$\text{deptID} \rightarrow \text{deptName}$

We decompose the relation into:

- `emp(empID, name, deptID)`
- `dept(deptID, deptName)`

	empID	name	deptID	deptName
Consider legal instances r :	1	Alice	10	CS
	2	Bob	10	CS
	3	Carol	20	Math

	empID	name	deptID
$\Pi_{\text{empID, name, deptID}}(r)$	1	Alice	10
	2	Bob	10
	3	Carol	20

	deptID	deptName
$\Pi_{\text{deptID, deptName}}(r)$	10	CS
	20	Math

	empID	name	deptID	deptName	
$\Pi_{\text{deptID, deptName}}(r) \bowtie \Pi_{\text{empID, deptName}}(r)$	1	Alice	10	CS	$= r$
	2	Bob	10	CS	
	3	Carol	20	Math	

Example (Lossy): Consider the relation: `R(empName, empLevel, empSalary)` with FDs:

$\text{empName} \rightarrow \text{empLevel}$

$\text{empName} \rightarrow \text{empSalary}$

$\text{empLevel} \rightarrow \text{empSalary}$

	empName	empLevel	empSalary
Consider legal instances r :	Alice	Junior	50K
	Bob	Junior	50K
	Carol	Senior	50K

	empName	empSalary
$\Pi_{\text{empName}, \text{empSalary}}(r)$	Alice	50K
	Bob	50K
	Carol	50K

	Level	Salary
$\Pi_{\text{empLevel}, \text{empSalary}}(r)$	Junior	50K
	Senior	50K

$\Pi_{\text{empName}, \text{empSalary}}(r) \bowtie \Pi_{\text{empLevel}, \text{empSalary}}(r)$

empName	empLevel	empSalary	
Alice	Junior	50K	$\neq r$
Alice	Senior	50K	
Bob	Junior	50K	
Bob	Senior	50K	
Carol	Junior	50K	
Carol	Senior	50K	

Note

A decomposition of $R(A, B, C)$ into $R_1(A, B)$ and $R_2(B, C)$ is lossless-join iff B is a **superkey** in R_1 or R_2

Dependency-Preserving: We call a decomposition dependency preserving if the *union* of FDs from decomposed relations yields all FDs in the original relation.

Example: If we decompose $R(\text{empName}, \text{empLevel}, \text{empSalary})$ into $R_1(\text{empName}, \text{empLevel})$, $R_2(\text{empName}, \text{empSalary})$. This is a **lossless-join** decomposition:

	empName	empLevel
$\Pi_{\text{empName}, \text{empLevel}}(r)$	Alice	Junior
	Bob	Junior
	Carol	Senior

	empName	Salary
$\Pi_{\text{empName}, \text{empSalary}}(r)$	Alice	50K
	Bob	50K
	Carol	50K

$\Pi_{\text{empName}, \text{empLevel}}(r) \bowtie \Pi_{\text{empName}, \text{empSalary}}(r)$

empName	empLevel	empSalary	
Alice	Junior	50K	$= r$
Bob	Junior	50K	
Carol	Senior	50K	

However, This is not *dependency preserving*. Proof:

FDs of $R_1 = \{\text{empName} \rightarrow \text{empLevel}\}$

FDs of $R_2 = \{\text{empName} \rightarrow \text{empSalary}\}$

The union of both is $\{\text{empName} \rightarrow \text{empLevel}, \text{empName} \rightarrow \text{empSalary}\}$ which is not equal to $\{\text{empName} \rightarrow \text{empLevel}, \text{empName} \rightarrow \text{empSalary}, \text{empLevel} \rightarrow \text{empSalary}\}$ i.e. the FD $\text{empLevel} \rightarrow \text{empSalary}$ is **lost**!

Example (Lossless-join and Dependency Preserving Decomposition)

$R_1(empName, empLevel), R_2(empLevel, empSalary)$

- Lossless-join because **empLevel** is superkey in R_2
- Dependency Preserving because $F_1 \cup F_2 = \{empName \rightarrow empLevel, empLevel \rightarrow empSalary\}$ and *through transitivity* we get $\{empName \rightarrow empSalary\}$

The Chase Test to Check the Lossless-Join Property

Input:

- A relation schema $R = \{A_1, A_2, \dots, A_n\}$
- A set of functional dependencies F
- A decomposition $\mathcal{D} = \{R_1, R_2, \dots, R_k\}$

Goal: Determine whether the decomposition is **lossless-join** with respect to F .

Procedure:

Step 1. Initialize a Chase Table:

- Create a table with one row per sub-relation $R_i \in \mathcal{D}$, and one column per attribute $A_j \in R$.
- For each row i and attribute A_j :
 - If $A_j \in R_i$, set the cell to α_{A_j}
 - If $A_j \notin R_i$, set the cell to a unique symbol β_{iA_j}

Step 2. Apply Functional Dependencies:

- For each FD $X \rightarrow Y \in F$, and for each row:
 - If all attributes in X have the same symbol in the row (e.g., all equal to σ),
 - Then set each attribute in Y to that same symbol σ
- Repeat until no further changes occur in the table.

Step 3. Check for Losslessness:

- If any row contains only the original α_{A_j} symbols across all attributes, the decomposition is **lossless**.
- Otherwise, it is **lossy**.

Example (Lossless)

Let $R(A, B, C)$, with FDs $F = \{A \rightarrow B\}$, and decomposition:

$R_1(A, B), R_2(A, C)$

Initial Chase Table:

A	B	C
α_A	α_B	β_{1C}
α_A	β_{2B}	α_C

A	B	C
α_A	α_B	β_{1C}
α_A	α_B	α_C

Apply $A \rightarrow B$:

Since row 2 contains only α -symbols, the decomposition is **lossless**.

Example (Lossy)

Let $R(A, B, C)$, with FDs $F = \{A \rightarrow B\}$, and decomposition:

$$R_1(A, B), \quad R_2(B, C)$$

Initial Chase Table:

A	B	C
α_A	α_B	β_{1C}
β_{2A}	α_B	α_C

Apply $A \rightarrow B$: Only row 1 has $A = \alpha_A$, but $B = \alpha_B$ already \rightarrow no changes.

No row becomes all- α , so the decomposition is **lossy**.

1NF: A relation is in First Normal Form (1NF) iff the domain contains only atomic single values.

2NF

A relation is in Second Normal Form iff:

- It is in 1NF
- There is no *non-prime* attribute A such that $Y \rightarrow A \in F$, where Y is a proper subset of a candidate key K ($Y \subsetneq K$)

In other words, if all *candidate* keys consist of a single attribute, 2NF is guaranteed.

Example: Consider the relation EMP_PROJ(EmpID, EmpName, ProjID, ProjName, ProjLocation, Hours) With Candidate Key (EmpID, ProjID) and set of FDs:

{(EmpID, ProjID \rightarrow Hours),
 (EmpID \rightarrow EmpName),
 (ProjID \rightarrow ProjName, ProjLocation)}

EmpName is a *non-prime* attribute that is determined by $\text{EmpID} \subsetneq \{\text{EmpID}, \text{ProjID}\}$ Hence 2NF is violated. (the same applies for ProjName, ProjLocation)

Normalization This relation can be normalized to 2NF by decomposing it in a way so we do not have these *partial* dependencies on the primary key: EMP(EmpID, EmpName), PROJ(ProjID, ProjName, ProjLocation), EMP_PROJ(EmpID, ProjID, Hours)

3NF

A relation is in Third Normal Form iff it is in 2NF and for each FD $X \rightarrow Y$, one of the following statements holds:

- $X \rightarrow Y$ is trivial ($Y \subseteq X$)
- X is a **superkey**
- Every attribute $A \in Y - X$ is a *prime* attribute

Example: Consider the relation EMP_DEPT(EmpID, EmpName, EmpBD, EmpAddress, DeptID, DeptName, DeptMgrId) With FDs:

- $\text{EmpID} \rightarrow \text{EmpName}, \text{EmpAddress}, \text{EmpBD}, \text{DeptID}$
- $\text{DeptID} \rightarrow \text{DeptName}, \text{DeptMgrId}$

For FD1, EmpID is superkey (we are safe). For FD2 it is not trivial, DeptID is not superkey, and none of DeptName or DeptMgrId is prime. So this violates 3NF.

Normalization

We must remove transitive dependencies to normalize to 3NF.

- EMP(EmpID, EmpName, EmpBD, EmpAddress, DeptID)
- DEPT(DeptID, DeptName, DeptMgrId)

BCNF

A relation R is Boyce-Codd Normal Form (BCNF) iff for each FD $X \rightarrow Y$, one of the following statements holds:

- $Y \subseteq X$ (trivial)
- X is a *superkey*

Note: Relations that violate BCNF are rare and hard to find in practice. In real projects, normalizing to 3NF usually guarantee BCNF except for some rare cases.

Example: Assume we are building a system to manage land properties across different districts. Each property is assigned a globally unique property ID.

Locally, properties are identified by the lot number, which is only unique within a district. A property is uniquely identified by district and lot number.

Additionally, each property belongs to one area (each district has multiple areas)

Consider we model this relation as $\text{LOTS}(\text{PropertyID}, \text{District}, \text{LotNum}, \text{Area})$ with FDs:

- $\text{PropertyID} \rightarrow \text{District}, \text{LotNum}, \text{Area}$
- $\text{District}, \text{LotNum} \rightarrow \text{PropertyID}, \text{Area}$
- $\text{Area} \rightarrow \text{District}$

We have the following candidate keys:

- PropertyID
- $\text{District}, \text{LotNum}$

The relation is in 3NF because, for each FD:

- PropertyID is a super key (candidate means minimal super key)
- $\text{District}, \text{LotNum}$ is a super key (same reasoning)
- District is *prime* since it is part of $\text{District}, \text{LotNum}$

The relation is not in BCNF because FD3 violates that (Area is not a super key)

Normalization

To normalize to BCNF, we decompose in a way that Area is a super key in one of the relations. Because Area determines district , we make a separate relation for that with Area as our primary: $\text{AREA_DISTRICT}(\text{Area}, \text{District})$

The relation have to be: $\text{LOTS}(\text{PropertyID}, \text{LotNum}, \text{Area})$

Disadvantage: We lost $\text{District}, \text{LotNum} \rightarrow \text{PropertyID}, \text{Area}$

7 Graph Databases and Cypher Queries

Graph Data Model

A graph database represents data as a property graph $G = (V, E, \lambda_V, \lambda_E)$, where:

- V : set of nodes (entities)
- E : set of edges (relationships)
- λ_V : node properties
- λ_E : edge properties

Example Node and Edge

- Node: (a:Person {name: 'Alice', age: 30})
- Edge: (a)-[:FRIEND_OF {since: 2015}]->(b)

Graph DBs favor connected data and avoid joins by treating relationships as first-class citizens.

Advantages of Graph Databases

- Efficient Traversals: Constant time access to related nodes.
- Schema Flexibility: Easily adapt to changing requirements.
- Natural Modeling: Direct mapping of real-world relationships.
- Performance: Particularly efficient for deep, multi-hop queries.

Graph DB Instances and Schemata

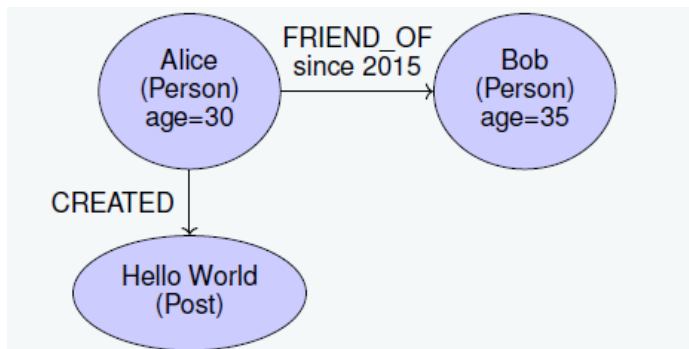


Figure 1: Instance

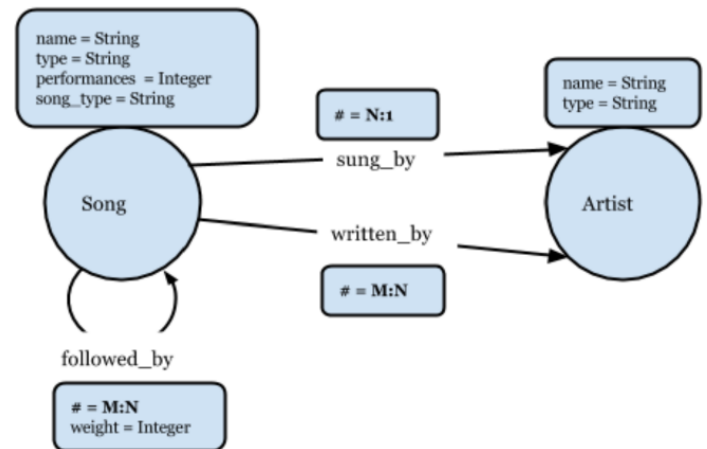


Figure 2: Schema

Note

Unlike relational databases, graph databases can be populated without prior specification of the schema!

Native vs Non-Native Storage

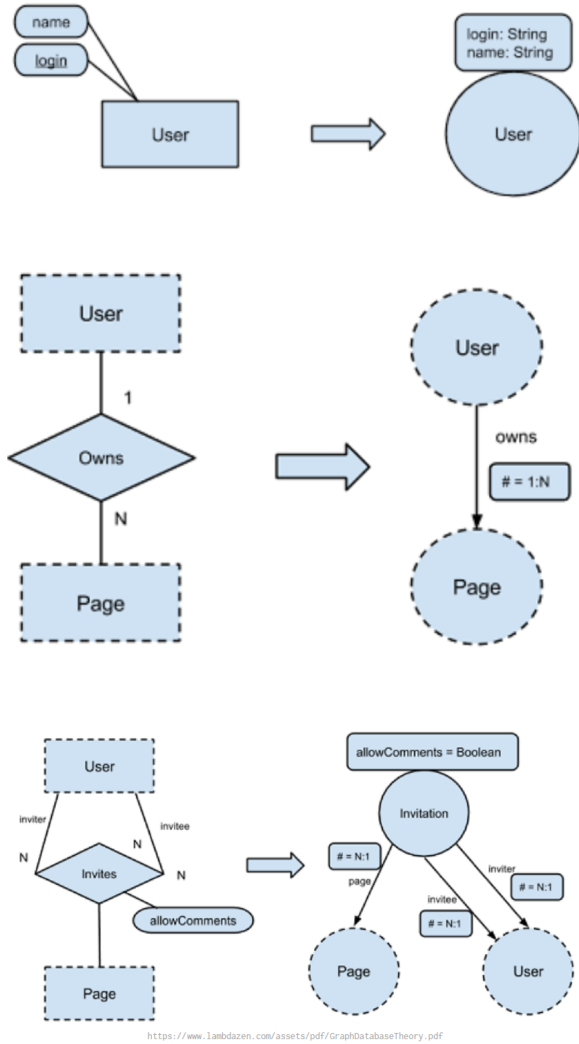
- **Native:** Index-free adjacency, fast for traversal
- **Non-Native:** Graph overlay on relational backend

ER to Graph Schema Transformation

- Entities \rightarrow Nodes
- Binary relations \rightarrow Edges

- N-ary relations \rightarrow Nodes with role-labeled edges

Examples:



Graph Schema Equivalence

Graph Universe: The graph universe $U(S)$ of a graph schema S is the infinite set of all graph instances of S

Equivalent: Two graph schemata S and \hat{S} are equivalent iff $\exists f.f : U(S) \rightarrow U(\hat{S})$

Graph Schema Transformation Rules

- **Renaming:** Change label/property names (*schema-preserving*)
- **Reverse Edges:** Invert direction if no conflict exists
- **Property Displacement:** Move edge prop to node if look-across is 1
- **Specialization/Generalization:** Split or merge types by property predicates
- **Edge Promotion:** Turn edge into node with two new edges
- **Property Promotion:** Factor property set into a separate node
- **Multivalued Expansion:** Turn list-valued property into connected nodes

Derived Vertex Types, Edge Types, and Properties

Let S be a graph schema and $U(S)$ the corresponding graph universe.

- A vertex type T is *derived* in S iff, for all graphs $G \in U(S)$, the graph G can be uniquely reconstructed from $G - V_T$, where V_T is the set of all vertices in G with type T .
- An edge type T is *derived* in S iff, for all graphs $G \in U(S)$, the graph G can be uniquely reconstructed from $G - E_T$, where E_T is the set of all edges in G with type T .
- A vertex or edge property P is *derived* in S iff, for all graphs $G \in U(S)$, the graph G can be uniquely reconstructed from a graph G' where the property P has been deleted from all vertices and edges.

General Rule To Simplify Graph Schemas

Given a graph schema S , deleting a derived property or vertex/edge type yields an equivalent graph schema \hat{S}

General Rule To Make Graph Schemas More Complex

Given a graph schema S , adding a vertex type, edge type, or property T such that T is derived in $\hat{S} = S + T$ yields an equivalent graph schema \hat{S}

Cypher Query Language

Cypher: Declarative graph query language used in Neo4j. Pattern-based, ASCII-art style.

Pattern Matching

```
MATCH (a:Person)-[:KNOWS]->(b) RETURN b.name
```

Core Constructs

- Node: `(p:Person {name: 'Anna'})`
- Edge: `[:KNOWS {since: 2019}]`
- Path: `(a)-[:KNOWS*1..3]-(b)` (1–3 undirected hops)
- Undirected Edge: `(a)-[:KNOWS]-(b)`

Modifying Data

CREATE

- `CREATE (a:Person {name: 'Bob'})` – new node
- `CREATE (a)-[:KNOWS]->(b)` – new edge

DELETE

- `DELETE a` – only if a has no edges
- `DETACH DELETE a` – delete node and connected edges

SET

- `SET a.surname = 'Smith'` – add or update property
- `SET a = {name: 'X', age: 42}` – overwrite all properties

Schema Constraints in Cypher

- **Uniqueness:** `REQUIRE (n.p1, n.p2) IS UNIQUE`
- **Existence:** `REQUIRE n.prop IS NOT NULL`
- **Type:** `REQUIRE n.prop IS :: STRING`
- **Key:** `REQUIRE (n.p1, n.p2) IS NODE KEY`

Example: Key Constraint: Ensure each Actor has a unique name combination:

```
CREATE CONSTRAINT actor_id FOR (a:Actor) REQUIRE (a.firstname, a.surname) IS NODE KEY
```


8 Descriptive Statistics and Data Normalization

Measures of Central Tendency

Mean: $\mu = \frac{1}{n} \sum_{i=1}^n x_i$

Note

Mean is best for symmetrical distributions without outliers

Optimization-Based Median

In the context of Optimization, the median for n numbers is a *set-valued function* when n is even and a *single-valued function* when n is odd. The function gives us the value or the set of values that minimizes the distance to all elements.

$$\text{Median} = \min_x \sum_{i=1}^n |x - x_i|$$

Examples (odd): data: $\{1, 2, 3, 6, 10\}$ we want to minimize $f(x) = \sum |x - x_i|$

- $f(1) = |1 - 1| + |1 - 2| + |1 - 3| + |1 - 6| + |1 - 10| = 0 + 1 + 2 + 5 + 9 = 17$
- $f(2) = |2 - 1| + |2 - 2| + |2 - 3| + |2 - 6| + |2 - 10| = 1 + 0 + 1 + 4 + 8 = 14$
- $f(3) = |3 - 1| + |3 - 2| + |3 - 3| + |3 - 6| + |3 - 10| = 2 + 1 + 0 + 3 + 7 = 13$
- $f(6) = |6 - 1| + |6 - 2| + |6 - 3| + |6 - 6| + |6 - 10| = 5 + 4 + 3 + 0 + 4 = 16$
- $f(10) = |10 - 1| + |10 - 2| + |10 - 3| + |10 - 6| + |10 - 10| = 9 + 8 + 7 + 4 + 0 = 28$

Hence, the minimum occurs at $x = 3$ with $f(3) = 13$. Since the number of data points is odd, the optimization-based median is **unique** and equal to the middle value in the sorted list: $\boxed{3}$.

Examples (even): data: $\{1, 2, 3, 6\}$ we want to minimize $f(x) = \sum |x - x_i|$

- $f(1) = |1 - 1| + |1 - 2| + |1 - 3| + |1 - 6| = 0 + 1 + 2 + 5 = 8$
- $f(2) = |2 - 1| + |2 - 2| + |2 - 3| + |2 - 6| = 1 + 0 + 1 + 4 = 6$
- $f(3) = |3 - 1| + |3 - 2| + |3 - 3| + |3 - 6| = 2 + 1 + 0 + 3 = 6$
- $f(6) = |6 - 1| + |6 - 2| + |6 - 3| + |6 - 6| = 5 + 4 + 3 + 0 = 12$

We observe that $f(x)$ attains its minimum value of 6 for any $x \in [2, 3]$. Since the number of data points is even, the optimization-based median is **set-valued**, and any value in the interval $\boxed{[2, 3]}$ minimizes the total absolute deviation.

Note

The Optimization-Based Median can be computed using simple rules when the data is sorted in ascending order and indexed as x_1, x_2, \dots, x_n . If n is odd, the median is the single value at index $i = \frac{n+1}{2}$ (i.e., the middle element). If n is even, the median is any value in the interval $[x_{n/2}, x_{\frac{n}{2}+1}]$.

Statistical Median

Conventionally in statistics, the median is always one single value, therefore when n is even, the median is calculated as $\frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$. and for odd we just use the middle value formula ($x_{\frac{n+1}{2}}$)

Example: Data: $\{1, 4, 7, 9\}$

Here, $n = 4$ is even, so we compute:

$$\text{median} = \frac{x_2 + x_3}{2} = \frac{4 + 7}{2} = \boxed{5.5}$$

Note

Median is preferred for skewed distributions or when outliers are present

Mode: The most frequent value(s) in the dataset. $\text{mode} = \arg \max_x |\{i \in \{1, \dots, n\} \mid x_i = x\}|$ **Example:**

Data: $\{2, 3, 5, 3, 8, 3, 2\}$

We compute the size of each set:

$$|\{i \mid x_i = 2\}| = 2 \quad (\text{indices 1 and 7})$$

$$|\{i \mid x_i = 3\}| = 3 \quad (\text{indices 2, 4, and 6})$$

$$|\{i \mid x_i = 5\}| = 1 \quad (\text{index 3})$$

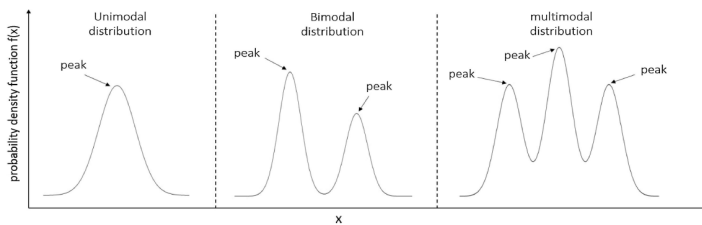
$$|\{i \mid x_i = 8\}| = 1 \quad (\text{index 5})$$

The maximum count is 3, which occurs at $x = 3$.

Note

Mode is useful for categorical data or to identify the most frequent value

Unimodal vs. Multimodal



Measures of Dispersion

Range: The difference between the maximum and minimum values ($\max_i X_i - \min_i X_i$)

Note

Range is best for a quick estimate of variability. However it is sensitive to outliers since it only considers extreme values

IQR: Interquartile Range (IQR) is the difference between the third quartile (Q_3) and first quartile (Q_1), measuring the spread of the middle 50% of data ($Q_3 - Q_1$)

Note

IQR is a robust measure of spread. It is useful when dealing with skewed distributions or outliers since it ignores extreme values

Variance σ^2 : The average squared deviation from the mean. Calculated as $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \mu_X)^2$

Unnormalized Variance (Total Variability): The raw sum of squares, called total sum of squares. $\text{TSS} = \sum_{i=1}^n (X_i - \mu_X)^2 = n\sigma_X^2$

Note

Variance measures overall data dispersion. However, squaring emphasizes larger deviations, making it more sensitive to extreme values

Standard Deviation σ : The square root of variance ($\sqrt{\sigma^2}$), indicating the average deviation from the mean

Unnormalized Standard Deviation: $\hat{\sigma}_X = \sqrt{\text{TSS}} = \sqrt{\sum_{i=1}^n (X_i - \mu_X)^2} = \sqrt{n\sigma_X^2} = \sqrt{n}\sigma_X$

Note

Standard Deviation is measured in the same unit as the data (more intuitive). It is used in many statistical methods like confidence intervals and hypothesis testing

Covariance: $\text{Cov}(X, Y) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)$

Unnormalized Covariance: $\text{Cov}_{\text{unnormalized}}(X, Y) = \sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y) = n\text{Cov}(X, Y)$

Coefficient of Variation (CV): relative measure of dispersion, calculated as the standard deviation divided by the mean ($\text{CV} = \frac{\sigma}{\mu} \times 100\%$)

Note

CV is useful for comparing variability across datasets with different units or scales. Low CV indicates less relative variability and more consistency. High CV suggests greater dispersion relative to the mean

Side Note: Only use CV for data measured on a ratio scale, where quantity ratios and zeros are meaningful.

Skewness: Measures data asymmetry by relating the average cubed distances from the mean to the standard deviation. It is calculated as $\frac{\frac{1}{n} \sum_{i=1}^n (X_i - \mu)^3}{\sigma^3}$.

The Cubed distances from the mean preserve sign of distances, tend to cancel out for symmetric distributions.

Note

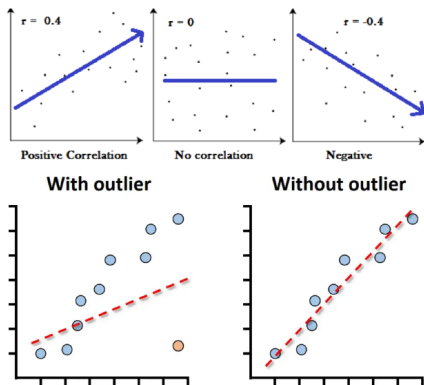
Skewness close to zero: indicates a symmetric distribution (mean, median and mode are close). Positive skewness: suggests a longer right tail (mode \leq median \leq mean). Negative skewness: indicates a longer left tail (mode \geq median \geq mean)

Measures of Correlation

Pearson's Correlation Coefficient

Assume we have quantitative data points $X = \{x_i \mid i \in [1 \dots n]\}$ and $Y = \{y_i \mid i \in [1 \dots n]\}$. Pearson's Correlation Coefficient (r) is defined as the covariance of X and Y divided by the product of their standard deviations:

$$r = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)}{\sigma_X \sigma_Y} = \frac{\text{Cov}(X, Y)}{\sqrt{\sigma_X^2} \sqrt{\sigma_Y^2}} = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu_X)^2} \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \mu_Y)^2}} \quad r(X, Y) \in [-1, 1]$$



Unnormalized r:

$$\hat{r} = \frac{\text{Cov}_{\text{unnormalized}}(X, Y)}{\sqrt{\text{TSS}_X} \sqrt{\text{TSS}_Y}} = \frac{\sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_{i=1}^n (x_i - \mu_X)^2} \sqrt{\sum_{i=1}^n (y_i - \mu_Y)^2}}$$

OLS Regression: To find the line that best fits our data and explain how our Y is linearly dependent on X , we find model parameters α and β that minimize the sum of squared errors $\sum_{i=1}^n (y_i - \hat{y}_i)^2$ where $\hat{y}_i = \alpha x_i + \beta + \epsilon_i$. There is a *closed-form* solution for this optimization problem and that is:

$$\alpha = \frac{\text{Cov}_{\text{unnormalized}}(X, Y)}{\text{TSS}_X} = \frac{\sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)}{\sum_{i=1}^n (x_i - \mu_X)^2}$$

$$= \frac{n\text{Cov}(X, Y)}{n\sigma_X^2} = \frac{\text{Cov}(X, Y)}{\sigma_X^2}$$

Remember: $r = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$. Meaning: $\text{Cov}(X, Y) = r\sigma_X \sigma_Y$

$$\alpha = \frac{r\sigma_X \sigma_Y}{\sigma_X^2} = r \frac{\sigma_Y}{\sigma_X}$$

$$\beta = \mu_Y - \alpha\mu_X$$

Explained Variability: Once we fit a line $\hat{y}_i = \alpha x_i + \beta + \epsilon$ we can calculate the explained variability. $\sum_{i=1}^n (\hat{y}_i - \mu_Y)^2$

Coefficient of Determination (R^2): What fraction of the total variability in Y is explained by our linear model.

$$R^2 = \frac{\text{Explained Variability}}{\text{Total Variability}} = \frac{\sum_{i=1}^n (\hat{y}_i - \mu_Y)^2}{\sum_{i=1}^n (y_i - \mu_Y)^2} \quad R^2 = 1 \rightarrow \text{perfect prediction}$$

Remember: $\hat{y}_i = \alpha x_i + \beta$ and $\mu_Y = \alpha\mu_X + \beta$

$$R^2 = \frac{\sum_{i=1}^n (\alpha x_i + \beta - (\alpha\mu_X + \beta))^2}{\sum_{i=1}^n (y_i - \mu_Y)^2}$$

$$R^2 = \frac{\sum_{i=1}^n (\alpha x_i + \beta - \alpha\mu_X - \beta)^2}{\sum_{i=1}^n (y_i - \mu_Y)^2}$$

$$R^2 = \frac{\sum_{i=1}^n (\alpha x_i - \alpha\mu_X)^2}{\sum_{i=1}^n (y_i - \mu_Y)^2}$$

$$R^2 = \frac{\sum_{i=1}^n (\alpha(x_i - \mu_X))^2}{\sum_{i=1}^n (y_i - \mu_Y)^2}$$

$$R^2 = \frac{\alpha^2 \sum_{i=1}^n (x_i - \mu_X)^2}{\sum_{i=1}^n (y_i - \mu_Y)^2} = \frac{\alpha^2 \text{TSS}_X}{\text{TSS}_Y} = \alpha^2 \frac{n\sigma_X^2}{n\sigma_Y^2} = \frac{\alpha^2 \sigma_X^2}{\sigma_Y^2}$$

Remember: $\alpha = r \frac{\sigma_Y}{\sigma_X}$ Hence $\alpha^2 = r^2 \frac{\sigma_Y^2}{\sigma_X^2}$

$$\begin{aligned} R^2 &= \alpha^2 \frac{\sigma_X^2}{\sigma_Y^2} \\ &= r^2 \frac{\sigma_Y^2}{\sigma_X^2} \frac{\sigma_X^2}{\sigma_Y^2} \\ R^2 &= r^2 \end{aligned}$$

Spearman's Rank Correlation Coefficient

Because r is sensitive to outliers and only suitable for linear relationships. We apply the Rank-Transformation trick to the data by the following steps:

1. Transform data $X = \{x_i \mid i \in [1 \dots n]\}$ and $Y = \{y_i \mid i \in [1 \dots n]\}$ into fractional ranks $X^R = \{x_i^R \mid i \in [1 \dots n]\}$ and $Y^R = \{y_i^R \mid i \in [1 \dots n]\}$
 - (a) For each x_i collect set $J_i = \{j \in [1 \dots n] \mid x_i = x_j\}$ of indices j of data points identical to x_i
 - (b) Sort the data in ascending order and store position π_i of x_i in sorted array (ordinal ranking)
 - (c) Compute fractional ranks as *mean* of ordinal ranks of identical values:

$$X_i^R = \frac{\sum_{i \in J_i} \pi_i}{|J_i|} = \frac{\min_{j \in J_i} \pi_i + \max_{j \in J_i} \pi_i}{2}$$

2. Compute Spearman's rank correlation coefficient as Pearson's correlation coefficient (r) of rank-transformed data X^R and Y^R

$$\rho = r(X^R, Y^R) = \frac{\sum_{i=1}^n (x_i^R - \mu_X^R)(y_i^R - \mu_Y^R)}{\sqrt{\sum_{i=1}^n (x_i^R - \mu_X^R)^2} \sqrt{\sum_{i=1}^n (y_i^R - \mu_Y^R)^2}}$$

Note

The mean of the ordinal ranks of elements in J_i is equal to the mean of the minimum and maximum of these ordinal ranks

Assume we have data $[1, 2, 2, 2, 3]$ and their ordinal ranks $[1, 2, 3, 4, 5]$. We have $k = 3$ tied elements starting from $r = 2$, their ordinal ranks are 2, 3, 4 which is $r, r+1, r+2$ and their mean is $\frac{r+(r+1)+(r+2)}{k}$. The general formula for the mean is:

$$\frac{(r) + (r+1) + (r+2) + \dots + (r+k-1)}{k}$$

And the mean of the first and last ordinal ranks is $\frac{r+(r+k-1)}{2} = \frac{2r+k-1}{2} = \frac{2r}{2} + \frac{k-1}{2} = r(\frac{2}{2}) + \frac{k-1}{2} = r + \frac{k-1}{2}$

We want to prove the following equality:

$$\frac{(r) + (r+1) + (r+2) + \dots + (r+k-1)}{k} = r + \frac{k-1}{2}$$

$$\frac{(r) + (r+1) + (r+2) + \dots + (r+k-1)}{k} = \frac{\sum_{i=0}^{k-1} r+i}{k} = \frac{1}{k} \sum_{i=0}^{k-1} r+i$$

$$\frac{1}{k} \sum_{i=0}^{k-1} r+i = \frac{1}{k} \left(\sum_{i=0}^{k-1} r + \sum_{i=0}^{k-1} i \right) \quad \sum_{i=0}^{k-1} r \text{ evaluates to } kr$$

$$\sum_{i=0}^{k-1} i = \frac{k(k-1)}{2} \quad \text{triangular numbers } \left(\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2} \right)$$

$$\frac{1}{k} \left(\sum_{i=0}^{k-1} r + \sum_{i=0}^{k-1} i \right) = \frac{1}{k} \left(kr + \frac{k(k-1)}{2} \right) = \frac{kr}{k} + \frac{k(k-1)}{2k} = r + \frac{k-1}{2}$$

Example:

Let:

$$X = [1, 1, 3, 5, 5, 5, 7], \quad Y = [2, 2, 4, 6, 7, 6, 9]$$

1. Rank-transform both X and Y **Step 1: Ordinal Ranks**

- Sorted X : $[1, 1, 3, 5, 5, 5, 7] \rightarrow$ Ordinal ranks: $[1, 2, 3, 4, 5, 6, 7]$
- Sorted Y : $[2, 2, 4, 6, 6, 7, 9] \rightarrow$ Ordinal ranks: $[1, 2, 3, 4, 5, 6, 7]$

Step 2: Fractional Ranks

- For X :

$$x_1 = x_2 = 1 \Rightarrow x_1^R = x_2^R = \frac{1+2}{2} = 1.5$$

$$x_3 = 3 \Rightarrow x_3^R = 3$$

$$x_4 = x_5 = x_6 = 5 \Rightarrow x_{4.6}^R = \frac{4+6}{2} = 5$$

$$x_7 = 7 \Rightarrow x_7^R = 7$$

- For Y :

$$y_1 = y_2 = 2 \Rightarrow y_1^R = y_2^R = \frac{1+2}{2} = 1.5$$

$$y_3 = 4 \Rightarrow y_3^R = 3$$

$$y_4 = y_6 = 6 \Rightarrow y_4^R = y_6^R = \frac{4+5}{2} = 4.5$$

$$y_5 = 7 \Rightarrow y_5^R = 6$$

$$y_7 = 9 \Rightarrow y_7^R = 7$$

Fractional Ranks:

$$X^R = [1.5, 1.5, 3, 5, 5, 5, 7] \quad Y^R = [1.5, 1.5, 3, 4.5, 6, 4.5, 7]$$

2. Compute Pearson r on ranks

- Mean ranks:

$$\mu_X^R = \frac{1}{7}(1.5 + 1.5 + 3 + 5 + 5 + 5 + 7) = \frac{28}{7} = 4$$

$$\mu_Y^R = \frac{1}{7}(1.5 + 1.5 + 3 + 4.5 + 6 + 4.5 + 7) = \frac{28}{7} = 4$$

- Numerator

$$\sum (x_i^R - \mu_X^R)(y_i^R - \mu_Y^R) \approx 22.25$$

- Denominator

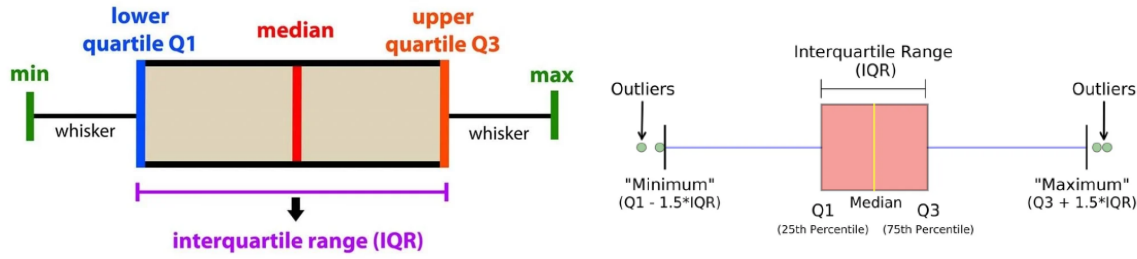
$$\sqrt{\sum_{i=1}^n (x_i^R - \mu_X^R)^2} \sqrt{\sum_{i=1}^n (y_i^R - \mu_Y^R)^2} = \sqrt{25.5} \sqrt{27}$$

- Then:

$$\rho = \frac{22.25}{\sqrt{25.5} \cdot \sqrt{27}} = \frac{22.25}{\sqrt{688.5}} \approx \frac{22.25}{26.24} \approx 0.848$$

Visualization

Box Plots



Use Case (Detecting Outliers): Given the sorted dataset:

`data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 500]`, $n = 10$

To compute the 25th percentile using linear interpolation, we use the formula:

$$\text{Position} = \frac{p}{100} \cdot (n - 1)$$

Substituting $p = 25$, we get:

$$\text{Position} = \frac{25}{100} \cdot (10 - 1) = 0.25 \cdot 9 = 2.25$$

This indicates that the 25th percentile lies 25% of the way between the values at positions 2 and 3 (using zero-based indexing), which are:

`data[2] = 30`, `data[3] = 40`

We linearly interpolate between these values:

$$\text{25th percentile} = 30 + 0.25 \cdot (40 - 30) = 30 + 2.5 = 32.5$$

Therefore, the 25th percentile of the dataset is:

$$Q1 = 32.5$$

For the 75th percentile: Substituting $p = 75$, we get:

$$\text{Position} = \frac{75}{100} \cdot (10 - 1) = 0.75 \cdot 9 = 6.75$$

This indicates that the 75th percentile lies 75% of the way between the values at positions 6 and 7, which are:

`data[6] = 70`, `data[7] = 80`

We linearly interpolate between these values:

$$\text{75th percentile} = 70 + 0.75 \cdot (80 - 70) = 70 + 7.5 = 77.5$$

Therefore, the 75th percentile of the dataset is:

$$Q3 = 77.5$$

Using $Q1$ and $Q3$, we can calculate $IQR = Q3 - Q1 = 77.5 - 32.5 = 45$

From that, we can calculate lower and upper bounds to detect outliers:

$$\text{Lower Fence} = Q1 - 1.5 \cdot IQR = 32.5 - 67.5 = -35$$

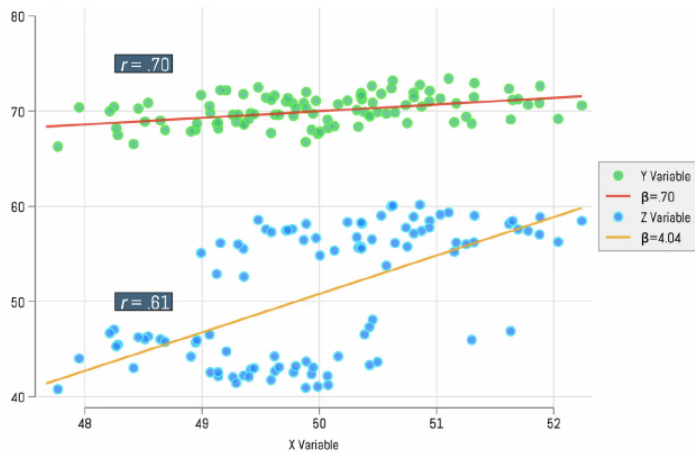
$$\text{Upper Fence} = Q3 + 1.5 \cdot IQR = 77.5 + 67.5 = 145$$

Min and Max (excluding outliers):

Min = 10, Max = 90

Outlier: The value 500 exceeds the upper fence and is therefore an outlier.

Scatter Plots



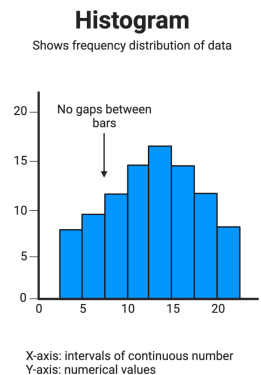
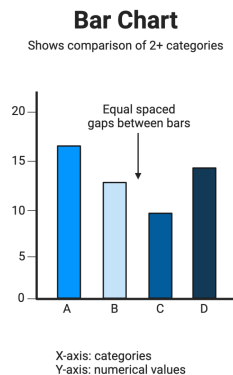
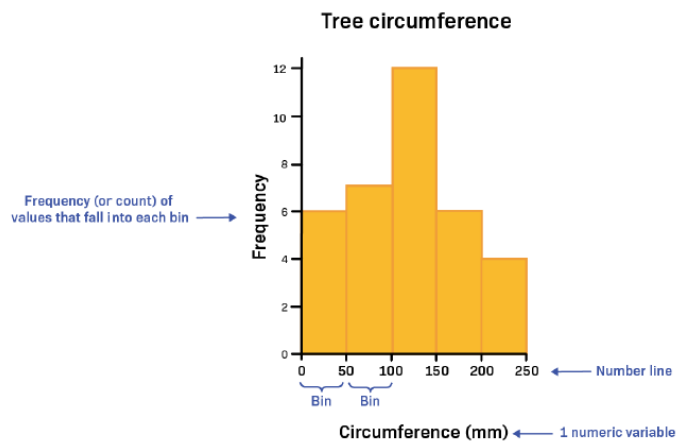
Heat Maps

Example of a color-coded heat map

A risk map offers a visualized, comprehensive view of the likelihood and impact of an organization's risks. Risks that fall into the green areas of the map require no action or monitoring. Yellow and orange risks require action. Risks that fall into red portions of the map need urgent action.

IMPACT	Catastrophic (5)	5	10	15	20	25
	Significant (4)	4	8	12	16	20
	Moderate (3)	3	6	9	12	15
	Low (2)	2	4	6	8	10
	Negligible (1)	1	2	3	4	5
		Improbable (1)	Remote (2)	Occasional (3)	Probable (4)	Frequent (5)
LIKELIHOOD						

Histograms and Bar Charts



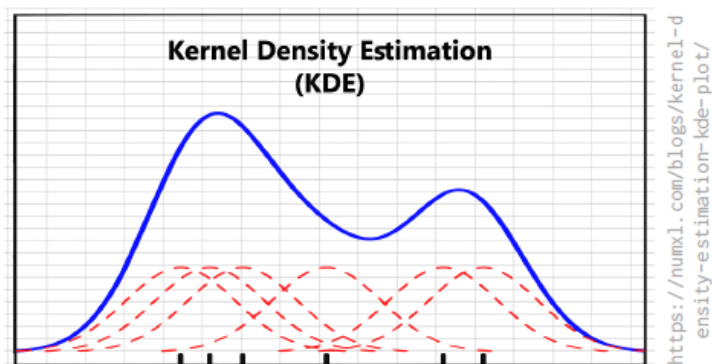
KDE Plots

- Provides smooth estimate of the probability density function (PDF) from data, as a continuous alternative to a histogram.

- Each data point x_i contributes a Gaussian $\mathcal{N}(x_i, h^2)$ to the total density. These are averaged to get an estimate of the full density

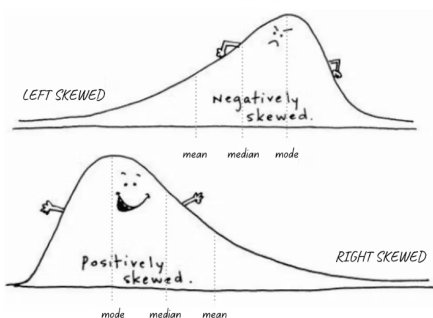
$$f(x) = \frac{1}{nh} \sum_{i=1}^n \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}$$

- too small $h \rightarrow$ under-smoothed plot
- too high $h \rightarrow$ over-smoothed plot



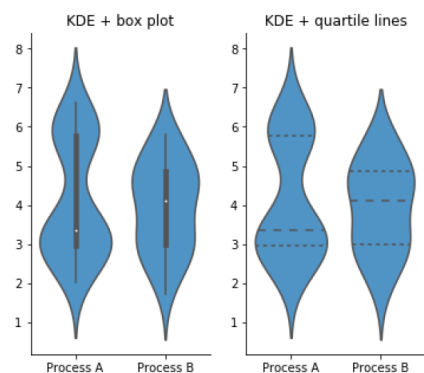
Detect skewness from density plots:

← Left or Right Skewed? Follow the tail. →



Violin Plots

A violin plot shows **summary statistics** (*box plot*) and **full distribution** (*KDE*)



Normalization

Min-Max

- Preserves orders and scaling. Fixed range $[0, 1]$ (Good for bounded data)
- Sensitive to outliers and does not generalize well

$$\hat{x} = \frac{x - \min}{\max - \min}$$

Z-Score

Useful when data is not bounded. Handles outliers well. Normalized data have zero mean and standard deviation of one. Handles varying distributions.

$$\hat{x} = \frac{x - \mu_X}{\sigma_X}$$

Robust Scaling

More resistant to outliers (best for handling outliers). Works well with skewed and non-Gaussian data

$$\hat{x} = \frac{x - \text{median}_X}{\text{IQR}_X}$$

Decimal Scaling

Used when you want to avoid using min-max but still need a fixed range (Quick and dirty)

$$\hat{x} = \frac{x}{10^k} \text{ where } k \text{ is the smallest natural number such that } \frac{x}{10^k} \in [-1, 1] \forall x$$

Log Transformation

- Reduces skewness and improves normality. Stabilizes variance and Preserves order.
- Best for reducing skewness in rightskewed data.
- Not defined for negative values and can distort small values

$$\hat{x} = \log(x + 1)$$

What to choose?

- No one-size-fits-all solution: choose based on data properties and requirement analysis
- Consider preprocessing techniques before normalization
- Always visualize data before and after transformation

Consider the Data Distribution

- If your data is *right-skewed*, use log transformation
- If it is with outliers, use robust scaling
- If transformed data needed in fixed range, then min-max or decimal
- If your data is normality distributed, then use Z-scores

9 Distance and Similarity Measures

Matrix Multiplication $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad AB = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$

Symmetric: A matrix A is symmetric iff $A = A^T$. Or iff $A_{ij} = A_{ji} \quad \forall i, j$. **Example:** $A = \begin{bmatrix} 2 & 5 \\ 5 & 3 \end{bmatrix}$

Positive Definite (PD): A **symmetric matrix** A is **positive definite** if, for every **nonzero** vector x , $x^T A x > 0$.

Example: Let $A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$ For any vector $x = \begin{bmatrix} a \\ b \end{bmatrix}$,

$$x^T A x = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 2a \\ 3b \end{bmatrix} = 2a^2 + 3b^2$$

For any $x \neq 0$, $2a^2 + 3b^2 > 0$, so A is positive definite.

Positive Semi-Definite (PSD): A **symmetric matrix** A is **positive semi-definite** if, for every vector x , $x^T A x \geq 0$.

Example:

Let $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ For any $x = \begin{bmatrix} a \\ b \end{bmatrix}$,

$$x^T A x = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} a \\ 0 \end{bmatrix} = a^2$$

$a^2 \geq 0$ always. So A is positive semi-definite, but not positive definite.

Gram Matrix: Given vectors x_1, x_2, \dots, x_n in \mathbb{R}^d , the **Gram matrix** G is the $n \times n$ matrix whose entries are all possible dot products: $G_{ij} = x_i^T x_j$ for $i, j = 1, \dots, n$. Example:

Let $x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $x_2 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$

Compute all dot products:

$$\begin{aligned} x_1^T x_1 &= 1 \times 1 + 2 \times 2 = 1 + 4 = 5 \\ x_1^T x_2 &= 1 \times 3 + 2 \times 4 = 3 + 8 = 11 \\ x_2^T x_1 &= 3 \times 1 + 4 \times 2 = 3 + 8 = 11 \\ x_2^T x_2 &= 3 \times 3 + 4 \times 4 = 9 + 16 = 25 \end{aligned}$$

Thus, the Gram matrix is

$$G = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 \\ x_2^T x_1 & x_2^T x_2 \end{bmatrix} = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix}$$

Note that, considering the vectors matrix $X = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$. The Gram matrix is $X^T X$

$$X^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad X^T X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix}$$

Properties of the Gram Matrix:

- Symmetric: $G_{ij} = G_{ji}$.
- Positive semi-definite (PSD): For any vector c , $c^T G c \geq 0$

Distance Measures: Quantify how far apart two objects are. *Properties:* symmetrical, non-negative, triangle inequality (direct path is at least as short as any detour), etc.

Similarity Measures: Quantify how similar or alike two objects are. *Properties:* symmetrical and ranges from 0 (completely different) to 1 (completely alike)

Metrics: Let X be a set of data objects, $x, y, z \in X$. A **metric** is a distance function $d : X \times X \rightarrow \mathbb{R}$ that satisfies:

1. Non-negativity: $d(x, y) \geq 0 \quad \forall x, y \in X$
2. Identity of indiscernibles: $d(x, y) = 0 \Leftrightarrow x = y$
3. Symmetry: $d(x, y) = d(y, x)$
4. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Pseudometrics: Same as **metrics** but allow zero distance between different objects

1. Non-negativity: $d(x, y) \geq 0 \quad \forall x, y \in X$
2. $d(x, x) = 0$ and it is possible that $d(x, y) = 0 \quad x \neq y$
3. Symmetry: $d(x, y) = d(y, x)$
4. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Example (Pseudometrics):

$$d(x, y) = \begin{cases} 0 & \text{if lowercase}(x) = \text{lowercase}(y) \\ 1 & \text{otherwise} \end{cases} \quad d(\text{Apple}, \text{apple}) = 0$$

Quasimetrics: A quasimetric is a distance function that *relaxes* the symmetry requirement of a **metric**

1. Non-negativity: $d(x, y) \geq 0 \quad \forall x, y \in X$
2. Identity of indiscernibles: $d(x, y) = 0 \Leftrightarrow x = y$
3. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

From Distance to Similarity and Vice Versa

Linear Scaling: $d(x, y) = 1 - \frac{s(x, y)}{s_{\max}}$ $s(x, y) = 1 - \frac{d(x, y)}{d_{\max}}$ Example: $d(\text{cat}, \text{cat}) = 0$ $s(\text{cat}, \text{cat}) = 1$

Note

Preserves relative differences in distances. Requires normalization of distances. Not suitable for unbounded distances.

Reciprocal transformation: $s(x, y) = \frac{1}{1+d(x, y)}$ Example: $d(\text{cat}, \text{dog}) = 2$ $s(\text{cat}, \text{dog}) = 0.33$

Note

Keeps values bounded between 0 and 1. Works well when small distances should lead to high similarity. Sensitive to large distances, as similarity first decays quickly and then slowly.

Exponential decay: $s(x, y) = e^{-\lambda d(x, y)}$ $\lambda = \frac{1}{\mu_d}$ Example:

- $d(\text{cat}, \text{cat}) = 0$
- $d(\text{cat}, \text{dog}) = 2$
- $d(\text{cat}, \text{lion}) = 4$
- $\mu_d = 2 \quad \lambda = 0.5$
- $s(\text{cat}, \text{cat}) = e^{-\lambda d(x, y)} = e^0 = 1$
- $s(\text{cat}, \text{dog}) = e^{-\lambda d(x, y)} = e^{-1} = 0.368$
- $s(\text{cat}, \text{lion}) = e^{-\lambda d(x, y)} = e^{-2} = 0.135$

Note

Smoothly decreases similarity with increasing distance. Handles large distances better than reciprocal transformation. Requires tuning λ and can be unintuitive when distances vary widely.

Dot Product: For vectors $x, y \in \mathbb{R}^n$ $|x| = |y|$ their dot product is given by $x \cdot y = \sum_{i=1}^n x_i y_i$

- Dot product between two vectors measure their **alignment**

- It ranges from $-\infty$ to $+\infty$
- If x and y point in the same direction, then $x \cdot y > 0$
- If x and y point in opposite directions, then $x \cdot y < 0$
- If they are **orthogonal (perpendicular)**, then $x \cdot y = 0$

Remember (The Norms):

A norm on a vector space is a function $\|\cdot\|$ that satisfies:

- Positive Definiteness: $\|x\| \geq 0$ with equality iff $x = 0$
- Homogeneity: $\|\lambda x\| = |\lambda| \|x\|$
- Triangle Inequality: $\|x + y\| \leq \|x\| + \|y\|$

Most Common Norms: L_1, L_2, L_p, L_∞

$$\|x\|_1 = \sum_{i=1}^n |x_i| \quad \|x\|_2 = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2} \quad \|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} \quad \|x\|_\infty = \max_i |x_i|$$

Note: a norm *induces* a **metric** $d(x, y) = \|x - y\|$ which satisfies:

- $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(z, y)$

Cosine Similarity: A normalized version of the **dot product**. It measures the angle between two vectors regardless their magnitude. $s_C(x, y) = \cos(\theta_{x,y}) = \frac{x \cdot y}{\|x\|_2 \|y\|_2}$

Note

Cosine similarity ranges from -1 (opposite direction) to 1 (same direction), if they are orthogonal then 0

Cosine Distance: $d_{\text{cosine}}(x, y) = 1 - s_C(x, y)$. Range is $[0, 2]$ (unintuitive)

Note

Cosine Distance does not satisfy the triangle inequality

Angular Distance: $d_{\text{angular}}(x, y) = \frac{\theta_{x,y}}{\pi} = \frac{\cos^{-1}(s_C(x,y))}{\pi}$. Range is $[0, 1]$

Note

Angular Distance satisfies the triangle inequality

Note

When comparing vectors of different magnitudes, neither cosine nor angular distance always respect the "identity of indiscernibles" axiom, meaning two different vectors can sometimes have zero distance under these measures if one is a scaled version of the other (colinear).

Example: Cosine Distance Does Not Satisfy Identity of Indiscernibles

Let $x = [1, 2]$ and $y = [2, 4]$ (note $y = 2x$):

$$\|x\|_2 = \sqrt{1^2 + 2^2} = \sqrt{5}$$

$$\|y\|_2 = \sqrt{2^2 + 4^2} = \sqrt{20}$$

$$x \cdot y = 1 \cdot 2 + 2 \cdot 4 = 10$$

$$s_C(x, y) = \frac{10}{\sqrt{5}\sqrt{20}} = \frac{10}{10} = 1$$

$$d_{\text{cosine}}(x, y) = 1 - 1 = 0$$

But $x \neq y$, so the identity of indiscernibles is not satisfied.

Example: Cosine Distance Violates the Triangle Inequality

Let $x = [1, 0]$, $y = [0, 1]$, and $z = [1, 1]$:

$$d_{\text{cosine}}(x, y) = 1 - 0 = 1$$

$$x \cdot z = 1 \cdot 1 + 0 \cdot 1 = 1$$

$$\|z\|_2 = \sqrt{1^2 + 1^2} = \sqrt{2}$$

$$s_C(x, z) = \frac{1}{1 \cdot \sqrt{2}} = \frac{1}{\sqrt{2}} \approx 0.707$$

$$d_{\text{cosine}}(x, z) = 1 - 0.707 = 0.293$$

Similarly, $d_{\text{cosine}}(y, z) \approx 0.293$

But:

$$1 \not\leq 0.293 + 0.293 = 0.586$$

So the triangle inequality is violated.

Example: Angular Distance Does Not Satisfy Identity of Indiscernibles

Let $x = [1, 2]$ and $y = [2, 4]$:

$$s_C(x, y) = 1$$

$$d_{\text{angular}}(x, y) = \frac{\arccos(1)}{\pi} = \frac{0}{\pi} = 0$$

But $x \neq y$, so the identity of indiscernibles is not satisfied.

Euclidean (L2) Distance: $d_{\text{Euclidean}}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$ (L2 Norm of the difference vector)

It is the most commonly used distance measure in vector spaces. It represents the straight-line (geometric) distance between two points in Euclidean space.

Note

Euclidean Distances satisfies all four metric properties. It is widely used in machine learning, clustering, nearest-neighbor search, etc., due to its intuitive geometric interpretation.

Manhattan (L1) Distance (taxicab): $d_{\text{Manhattan}}(x, y) = \sum_{i=1}^n |x_i - y_i|$ (L1 Norm of the difference vector)

It measures the shortest path between points if movement is restricted to axis-aligned steps (like navigating city blocks in a grid). It satisfies all four metric properties. It is often used in ML and clustering when axis-aligned movement makes sense.

Minkowski (L_p) Distance: $d_p(x, y) = (\sum_{i=1}^n |x_i - y_i|^p)^{1/p}$

Note

The larger p , the bigger the impact of the dimensions with the larger differences on the overall distance. The Minkowski distance satisfies all metric properties $\forall p \geq 1$

Chebyshev (L_∞) Distance: $d_\infty(x, y) = \max_i |x_i - y_i|$

- Measures the greatest absolute difference between any coordinate of two vectors
- It is the limit case of the Minkowski distance as $p \rightarrow \infty$
- Useful for scenarios where only the largest coordinate difference matters
- The unit ball (set of points at distance 1) under Chebyshev distance is a square in 2D

Remember: Unit Ball

For a given norm (distance function) $\|\cdot\|$, the unit ball is the set of all points (vectors) whose distance from the origin is less than or equal to one. Formally, in \mathbb{R}^n : $B = \{x \in \mathbb{R}^n \mid \|x\| \leq 1\}$. For example, consider \mathbb{R}^2 and L2 Norm, B is the set of points (x_1, x_2) such that $\sqrt{(x_1)^2 + (x_2)^2} \leq 1$, i.e. $x_1^2 + x_2^2 \leq 1$.

Remember, the circle of radius r is the set of points (x_1, x_2) such that $x_1^2 + x_2^2 = r^2$. Hence, unit ball of L2 norm is a circle.

For a point (x_1, x_2) the Chebyshev norm is $\max(|x_1|, |x_2|)$. The unit ball for Chebyshev norm are the points (x_1, x_2) where $\max(|x_1|, |x_2|) \leq 1$. Hence, unit ball of L_∞ is a square.

For L_1 , the unit ball is all points (x_1, x_2) such that $|x_1| + |x_2| \leq 1$. Hence, the unit ball is a diamond for L_1 .

Excursion

Let's pick a vector and compute two different norms:

Let $x = (3, 4)$ in \mathbb{R}^2 .

$$\|x\|_1 = |3| + |4| = 7$$

$$\|x\|_2 = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Let's try another two vectors:

Let $y = (1, 0)$:

$$\|y\|_1 = |1| + |0| = 1$$

$$\|y\|_2 = \sqrt{1^2 + 0^2} = 1$$

Let $z = (1, 1)$:

$$\|z\|_1 = |1| + |1| = 2$$

$$\|z\|_2 = \sqrt{1^2 + 1^2} = \sqrt{2} \approx 1.414$$

Compare the Norms:

Vector	$\ x\ _1$	$\ x\ _2$	$\ x\ _2 \leq \ x\ _1$	$\ x\ _1 \leq \sqrt{2}\ x\ _2$
(3, 4)	7	5	$5 \leq 7$	$7 \leq 7.07$
(1, 0)	1	1	$1 \leq 1$	$1 \leq 1.41$
(1, 1)	2	1.414	$1.414 \leq 2$	$2 \leq 2$

In all cases,

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{2}\|x\|_2$$

The General Pattern: For any vector x in \mathbb{R}^2 , $\|x\|_2 \leq \|x\|_1 \leq \sqrt{2}\|x\|_2$. For x in \mathbb{R}^n , $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$

In the general equivalence of norms, C_1 and C_2 are the constants making the inequalities true for all vectors:

$$C_1\|x\|_2 \leq \|x\|_1 \leq C_2\|x\|_2$$

For L_1 and L_2 in \mathbb{R}^2 , these are $C_1 = 1$ and $C_2 = \sqrt{2}$:

$$1 \cdot \|x\|_2 \leq \|x\|_1 \leq \sqrt{2} \cdot \|x\|_2$$

Theorem: For any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on a finite-dimensional vector space, there exist constants $C_1, C_2 > 0$ such that, for all vectors x :

$$C_1\|x\|_a \leq \|x\|_b \leq C_2\|x\|_a$$

Correlation-Based Similarity

- Pearson r quantifies similarity as degree of linear correlation
- Spearman ρ quantifies similarity as monotonicity of x w.r.t y
- $\text{corr}(x, y) \in [-1, 1]$

Kernel Function: A function $k(x, \bar{x})$ that computes the similarity between two input vectors x and \bar{x} and satisfies the following properties

1. Symmetry: $k(x, \bar{x}) = k(\bar{x}, x)$
2. Positive semi-definiteness: For any finite set $\{x_1, \dots, x_n\}$ of vectors, the Gram matrix $K = (K_{ij}) \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = k(x_i, x_j)$ is positive semi-definite (PSD), i.e. satisfies

$$c^T K c = \sum_{i=1}^n \sum_{j=1}^n c_i c_j k(x_i, x_j) \leq 0$$

for all vectors of coefficients $c = (c_i) \in \mathbb{R}^n$

Note

The definition of a kernel does not require that k is non-negative.

Linear Kernel: The simplest form of kernel (dot product similarity). $k(x, \bar{x}) = x \cdot \bar{x} = x^T \bar{x}$

Example: consider $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ Then $k(x, \bar{x}) = x^T \bar{x} = 1 \cdot 3 + 2 \cdot 4 = 3 + 8 = 11$

Proof that linear kernel is PSD:

For input vectors x_1, x_2, \dots, x_n our vectors matrix is $X = [x_1, x_2, \dots, x_n]$ and the Gram matrix $K = X^T X$ (as shown previously)

For any vector c , we have $c^T K c = c^T X^T X c = (Xc)^T (Xc) = \|Xc\|_2^2 \geq 0 \quad \forall c$

Gaussian (RBF) Kernel: The Gaussian (radial basis function) kernel is the most widely used kernel in the ML field.
 $k(x, \bar{x}) = \exp\left(-\frac{\|x - \bar{x}\|_2^2}{2\sigma^2}\right)$

- small $\sigma \rightarrow$ kernel is more sensitive to distance between vectors
- large $\sigma \rightarrow$ kernel is less sensitive to distance between vectors

Polynomial Kernel: $k(x, \bar{x}) = (x^T \bar{x} + c)^d$. Where $d \geq 1$ is the degree. If $d = 1$, the kernel captures linear relationships between the variables, if $d = 2$, the kernel captures quadratic relationships, and so on.

Note

The larger the constant $c \geq 0$ the smoother the similarity landscape (small differences get swallowed by c)

Example: let $c = 1$ and $d = 2$, and assume we have 1D data $x, y \in \mathbb{R}$ then $k(x, y) = (xy + 1)^2 = x^2 y^2 + 2xy + 1$

Notice that $x^2 y^2 + 2xy + 1$ is the result of dot product between two feature vectors $\phi(x) = [x^2, \sqrt{2}x, 1]$ and $\phi(y) = [y^2, \sqrt{2}y, 1]$

Which means that the kernel is *implicitly* mapping our original 1D data (x, y) into a 3D space. i.e., we can instead of transforming all data into higher dimensions, we just use the kernel function which gives us inner product result as if we had transformed.

- A linear kernel only detects linear alignment between data points
- A Polynomial kernel defines similarity in a much richer way: it says two inputs are similar if their higher-order features are similar, not just their values.

Mercer's Theorem: if k is a symmetric and PSD kernel function, then there exists a feature map ϕ into a higher-dimensional space D (possibly ∞) such that $k(x, y) = \phi(x) \cdot \phi(y)$

Note

In these higher-dimensional spaces, patterns may be detectable that are impossible or difficult to individuate in the original feature spaces.

Kernel Trick: A method in machine learning that enables algorithms which depend only on inner products between data points to be generalized to nonlinear feature spaces.

Note

The kernel trick allows algorithms to operate in high- or infinite-dimensional feature spaces at the computational cost of operating in the original space, thereby enabling them to learn nonlinear patterns while remaining scalable.

Applications of the Kernel Trick in Machine Learning:

- Support vector machines (SVMs): Supervised ML approach that implicitly fits separating hyperplanes in the higher-dimensional spaces.
- Kernelized perceptron: Another supervised ML approach that uses the kernel trick to learn non-linear decision boundaries.
- Kernel principal component analysis (kernel PCA): An extension of PCA that uses kernel functions to perform non-linear dimensionality reduction. It implicitly maps the data to a higher-dimensional space and then performs PCA in that space, allowing for the extraction of non-linear principal components.
- Kernelized clustering (e.g., k-means): Clustering methods that use the kernel trick to implicitly compute distances in the higher-dimensional space, allowing for the identification of non-linear clusters in the original data space.

Edit Distances: Ways to measure how different two structured objects are, typically strings, trees, or graphs.

- You define a set of basic operations that can “edit” one object into another. *Examples for strings:* insert a character, delete a character, substitute one character for another.
- Each operation has a cost (often 1 for each, but could be different).
- The edit distance between two objects S and T is the minimum total cost needed to turn S into T by a sequence of allowed operations.

Edit distances always respect the triangle inequality. If you know a way to turn A into B with cost $d(A, B)$, and a way to turn B into C with $d(B, C)$, then you could do both sequences of edits to turn A into C at a total cost $d(A, B) + d(B, C)$

Edit distances are generally not symmetric (Quasimetrics). The cost to edit A into B is not always the same as the cost to edit B into A .

Levenshtein Distance (metric): An edit distance for strings, defined as minimum number of single-character insertions, deletions, or substitutions needed to transform a source string a into a target string b .

Base Cases:

$$\text{lev}(a, "") = |a|$$

$$\text{lev}("", b) = |b|$$

Recursive Cases:

$$\text{lev}(a, b) = \begin{cases} \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if head}(a) = \text{head}(b) \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{if head}(a) \neq \text{head}(b) \end{cases}$$

Example:

$$\text{lev}(\text{"kit"}, \text{"cat"}) = \text{lev}(\text{"ki"}, \text{"ca"}) \quad (\text{heads: 't' = 't'})$$

$$\text{lev}(\text{"ki"}, \text{"ca"}) = 1 + \min \begin{cases} \text{lev}(\text{"k"}, \text{"ca"}) \\ \text{lev}(\text{"ki"}, \text{"c"}) \\ \text{lev}(\text{"k"}, \text{"c"}) \end{cases} \quad (\text{heads: 'i' } \neq \text{'a'})$$

$$\text{lev}(\text{"k"}, \text{"ca"}) = 1 + \min \begin{cases} \text{lev}(\text{"", "ca"}) = 2 \\ \text{lev}(\text{"k"}, \text{"c"}) = 1 \\ \text{lev}(\text{"", "c"}) = 1 \end{cases} \quad (1 + \min\{1, 1, 0\})$$

$$\text{lev}(\text{"ki"}, \text{"c"}) = 1 + \min \begin{cases} \text{lev}(\text{"k"}, \text{"c"}) = 1 \\ \text{lev}(\text{"ki"}, \text{""}) = 2 \\ \text{lev}(\text{"k"}, \text{""}) = 1 \end{cases}$$

$$\text{lev}(\text{"k"}, \text{"c"}) = 1 + \min \begin{cases} \text{lev}(\text{"", "c"}) = 1 \\ \text{lev}(\text{"k"}, \text{""}) = 1 \\ \text{lev}(\text{"", ""}) = 0 \end{cases}$$

Finally,

$$\text{lev}(\text{"ki"}, \text{"ca"}) = 1 + \min\{2, 2, 1\} = 1 + 1 = 2$$

$$\text{lev}(\text{"kit"}, \text{"cat"}) = \text{lev}(\text{"ki"}, \text{"ca"}) = 2$$

Graph Edit Distance (GED): Given two graphs G_1 and G_2 , $\text{GED}(G_1, G_2)$ is minimum cost of a sequence of edit operations that transform G_1 into a graph \tilde{G}_2 that is **isomorphic** to G_2

GED Edit Operations: There are six elementary edit operations: Insertions of isolated nodes, deletions of isolated nodes, attribute substitutions for nodes, deletions of edges, insertions of edges between existing nodes, attribute substitutions for edges.

Note

If "identity" means "isomorphism", then GED satisfies all pseudometric axioms. If "identity" means exact equality, then GED would be a metric.

Shortest-Path Distance: Let $G = \langle V, E \rangle$ be a graph, and let $w : E \rightarrow \mathbb{R}_{\geq 0}$ assign a non-negative weight (cost) to each edge. The shortest-path distance from node u to $v \in V$ is: $d_G(u, v) = \min_P \sum_{e \in P} w(e)$

In other words: The shortest-path distance between two nodes is the smallest possible total cost to travel from u to v along the edges of the graph.

Note

In undirected graphs with symmetric edge costs, Shortest-path distance is a metric. In directed graphs or graphs with asymmetric edge costs, Shortest-path distance is a quasimetric.

Linkage Distances for Clusters of Data Objects

Consider we have a set of data objects V that are grouped into n clusters $\{C_1, \dots, C_n\}$. And we also have a distance measure between individual objects $u, v \in V$ defined as $d_V : V \times V \rightarrow \mathbb{R}$.

We want to define a distance measure between **clusters** ($d(C_i, C_j)$)

Linkage Distances: Ways to aggregate (combine) the distances between all pairs of points in two clusters, in order to get a single distance value between the clusters themselves.

Single Linkage: $d_{SL}(C_i, C_j) = \min\{d_V(u, v) \mid (u, v) \in C_i \times C_j\}$

For all possible pairs of points (u, v) where $u \in C_i$ and $v \in C_j$ compute the distance $d_V(u, v)$ and take the smallest of these values. This is the minimum distance between the two clusters.

Complete Linkage: $d_{CL}(C_i, C_j) = \max\{d_V(u, v) \mid (u, v) \in C_i \times C_j\}$ (maximum distance)

Average Linkage: $d_{AL}(C_i, C_j) = (|C_i| \cdot |C_j|)^{-1} \sum_{u \in C_i} \sum_{v \in C_j} d_V(u, v)$ (mean distance) i.e. for all possible pairs (u, v) , sum all the distance and divide by number of all possible pairs.

Sets Similarity and Distance

Jaccard Similarity: For sets A, B , the Jaccard similarity is given by $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ range $\rightarrow [0, 1]$

Jaccard Distance (metric): $d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$

Sørensen-Dice Similarity: $S(A, B) = \frac{2|A \cap B|}{|A| + |B|}$. It is never smaller than the Jaccard index, $S(A, B) \geq J(A, B)$

Sørensen-Dice Distance: $d_S(A, B) = 1 - \frac{2|A \cap B|}{|A| + |B|}$

Note

The Sørensen-Dice “distance” fails the triangle inequality, so it is not a metric.

Proof by example:

Let:

$$A = \{1\}, \quad B = \{1, 2\}, \quad C = \{2\}$$

Compute the Sørensen-Dice similarity for each pair:

$$S(A, B) = \frac{2|A \cap B|}{|A| + |B|} = \frac{2 \cdot 1}{1 + 2} = \frac{2}{3}$$

$$S(B, C) = \frac{2|B \cap C|}{|B| + |C|} = \frac{2 \cdot 1}{2 + 1} = \frac{2}{3}$$

$$S(A, C) = \frac{2|A \cap C|}{|A| + |C|} = \frac{2 \cdot 0}{1 + 1} = 0$$

Compute the corresponding distances:

$$d_S(A, B) = 1 - S(A, B) = 1 - \frac{2}{3} = \frac{1}{3}$$

$$d_S(B, C) = 1 - S(B, C) = 1 - \frac{2}{3} = \frac{1}{3}$$

$$d_S(A, C) = 1 - S(A, C) = 1 - 0 = 1$$

Sum the distances for the triangle inequality:

$$d_S(A, B) + d_S(B, C) = \frac{1}{3} + \frac{1}{3} = \frac{2}{3}$$

But:

$$d_S(A, C) = 1 > \frac{2}{3} = d_S(A, B) + d_S(B, C)$$

Binary Data

Iverson bracket: $[\text{True}] = 1$ and $[\text{False}] = 0$

Hamming Distance (metric): Quantifies the bit-wise difference between two binary arrays of the same size. For two binary arrays $A, B \in \{0, 1\}^n$ it is defined as $\text{HD}(A, B) = \frac{1}{n} \sum_{i=1}^n [A_i \neq B_i]$