Data Visualization Final Report

Hai Ngoc NGUYEN

December 22, 2024

1 Practical Work 1

1.1 Exercise 1

The visual representation demonstrates:

Vertex set: $V = v_1, v_2, v_3, v_4, v_5$ where $v_1 = a$, $v_2 = b$, $v_3 = c$, $v_4 = d$, $v_5 = e$
Edge set: $E = e_1, e_2, e_3, e_4$ where:

$e_1 = (a, c)$ $e_2 = (a, d)$ $e_3 = (d, e)$ $e_4 = (b, d)$

The graph shows a clear hierarchical structure with:

Level 1: vertices a, b Level 2: vertices c, d Level 3: vertex e

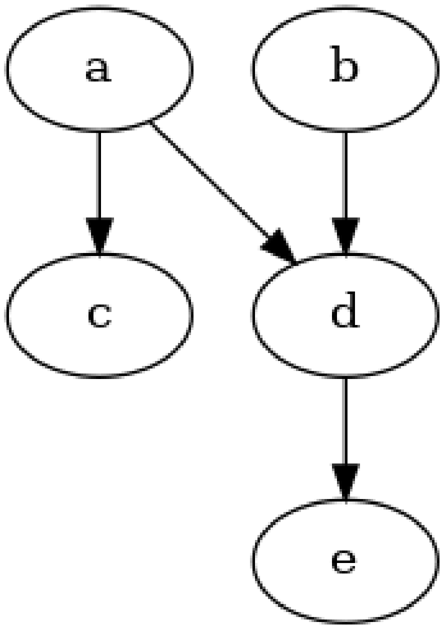


Figure 1: Exercise 1

1.2 Exercise 2

Let $G = (V, E)$ be a directed graph with attributes where: $V = a, b, c, d, e$ is the vertex set with attributes:

$\text{attr}(a) = \text{color} : \text{red}\ \text{attr}(b) = \text{color} : \text{blue}\ \text{attr}(c) = \text{color} : \text{green}\ \text{attr}(d) = \text{color} : \text{yellow}\ \text{attr}(e) = \text{color} : \text{orange}$

$E = (a, c), (a, d), (b, d), (d, e)$ with edge attributes:

$\text{attr}(a, c) = \text{label} : \text{"Step 1"}, \text{color} : \text{purple}\ \text{attr}(a, d) = \text{label} : \text{"Step 2"}, \text{color} : \text{brown}\ \text{attr}(b, d) = \text{label} : \text{"Step 3"}, \text{color} : \text{orange}\ \text{attr}(d, e) = \text{label} : \text{"Step 4"}, \text{color} : \text{black}$

The DOT representation demonstrates:

Node attributes using square brackets: [label="...", color=..., style=filled] Edge attributes using square brackets: [label="...", color=...] Visual hierarchy preserved from Exercise 1 with enhanced visual properties:

Level 1: a (red), b (blue) Level 2: c (green), d (yellow) Level 3: e (orange)

Edge progression is clearly marked through sequential steps: $a \xrightarrow{\text{Step 1}} c$ $a \xrightarrow{\text{Step 2}} d$ $b \xrightarrow{\text{Step 3}} d$ $d \xrightarrow{\text{Step 4}} e$ This visualization enhances the basic graph structure with visual attributes for both nodes and edges, making the graph more informative and visually distinct.

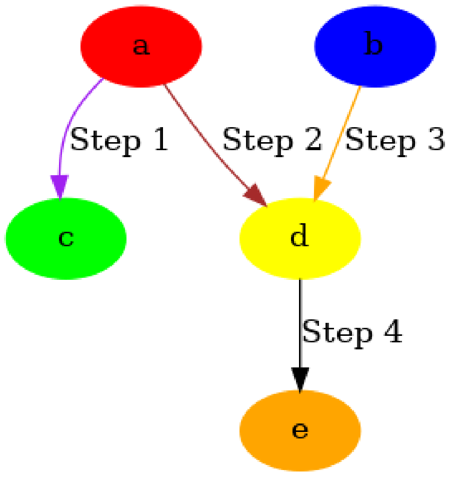


Figure 2: Exercise 2

1.3 Exercise 3

Consider a random graph $G = (V, E)$ generated using the $G_{n,m}$ random graph model: $G_{n,m} = (V_n, E_m)$ where:

$|V| = n = 10$ vertices $|E| = m = 20$ edges Properties of $G_{n,m}$:

Vertex set: $V = v_1, v_2, ..., v_{10}$ Edge probability:

$P(e_{ij}) = \frac{m}{\binom{n}{2}} = \frac{20}{\binom{10}{2}} \approx 0.44$

Graph metrics:

Average degree: $\bar{d} = \frac{2|E|}{|V|} = \frac{2m}{n} = \frac{2(20)}{10} = 4$ Graph density: $\rho = \frac{2|E|}{|V|(|V|-1)} = \frac{2m}{n(n-1)} = \frac{2(20)}{10(9)} \approx 0.44$

Layout algorithm:

Spring layout energy function: $E = \sum_{(i,j)\in E} k_s(d_{ij} - \ell)^2 + \sum_{i\neq j} \frac{k_r}{d_{ij}^2}$ where: d_{ij} is the distance between vertices i and j k_s is the spring constant k_r is the repulsion constant ℓ is the natural spring length

The visualization shows:

Vertices V as labeled blue circles Edges E as straight lines Force-directed layout for optimal vertex positioning

This random graph generation demonstrates fundamental concepts in graph theory while providing a clear visual representation of the resulting structure.

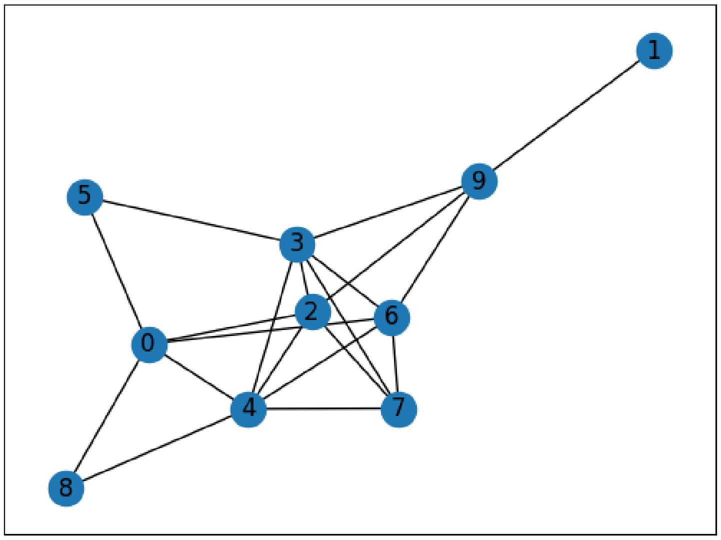


Figure 3: Exercise 3

1.4 Exercise 4

Consider directed graph $G = (V, E)$ read from DOT file: $V = a, b, c, d, e$ $E = (a, c), (a, d), (b, d), (d, e)$ Visualization parameters:

Node attributes: $\phi_{node} = \text{color} : \text{lightgreen}, \text{size} : 800$ Edge attributes: $\phi_{edge} = \text{arrows} : \text{True}$ Plot size: $\phi_{fig} = (8, 6)$

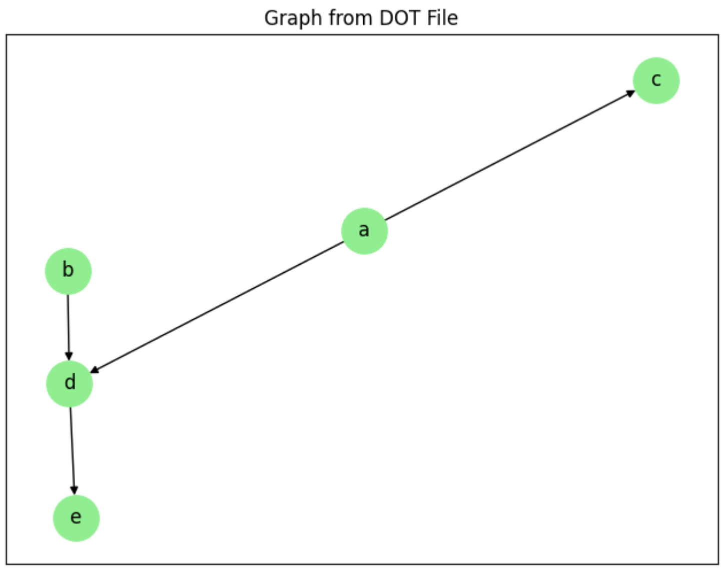


Figure 4: Exercise 4

1.5 Exercise 5

Consider a divisor graph $G = (V, E)$ where: $V = 2, 3, ..., n$, where $n = 10$ $E = (i, j)|i, j \in V, i \bmod j = 0$ The edge set can be expanded as:

$(4, 2)$: 4 divides by 2 $(6, 2), (6, 3)$: 6 divides by 2,3 $(8, 2), (8, 4)$: 8 divides by 2,4 $(9, 3)$: 9 divides by 3 $(10, 2), (10, 5)$: 10 divides by 2,5 Visualization parameters:

$\phi_{node} = \text{color : skyblue, size : 800}$ $\phi_{edge} = \text{arrows : True}$ The resulting graph shows the divisibility relationships between integers from 2 to 10.

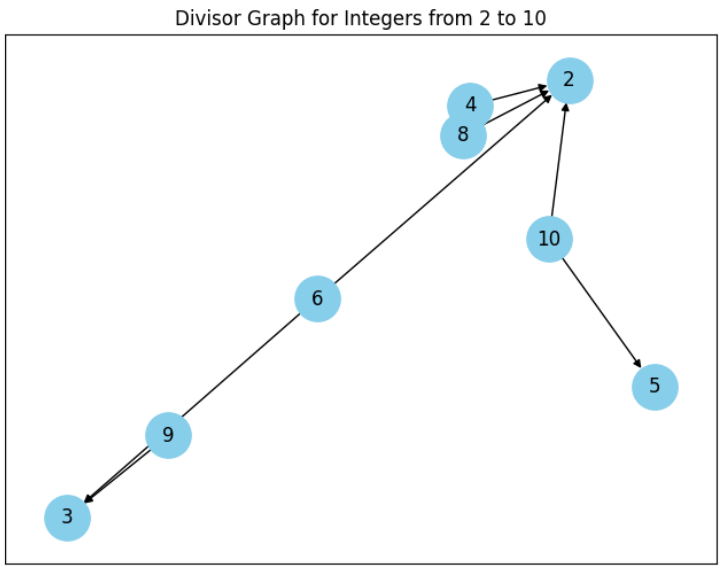


Figure 5: Exercise 5

1.6 Exercise 6

Function definition: $\text{mydraw}(G, E_{color})$ where:

$G = (V, E)$ is a directed graph $E_{color} \subseteq E$ is the subset of edges to highlight

Graph parameters:

$G = (V, E)$ where:

$V = 1, 2, 3, 4$ $E = (1, 2), (2, 3), (3, 4), (1, 3), (4, 2)$

$E_{color} = (1, 3), (3, 4)$

Visualization algorithm:

Compute layout: $\text{pos} = f_{spring}(G)$ Base graph drawing: $\text{draw}(G, \phi_{base})$ where:

$\phi_{base} = \text{color} : \text{lightblue}, \text{size} : 800$

Highlight edges: $\text{draw}(G, E_{color}, \phi_{highlight})$ where:

$\phi_{highlight} = \text{color} : \text{red}, \text{width} : 2.5$

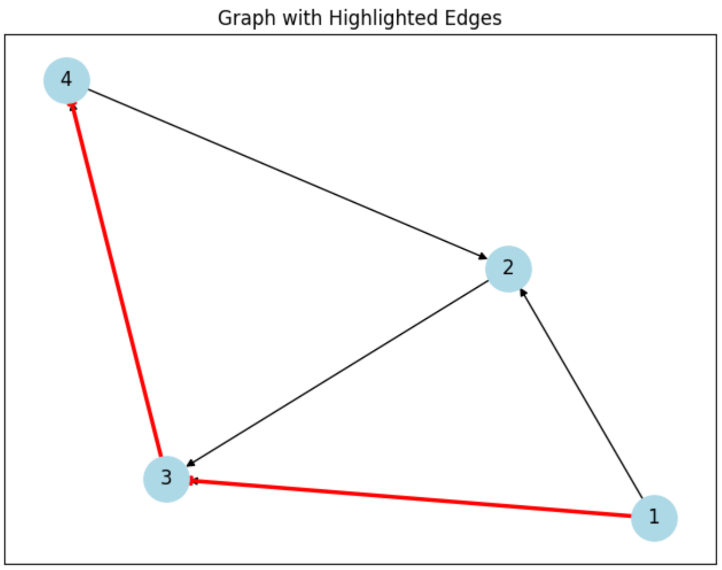


Figure 6: Exercise 6

1.7 Exercise 7

Let $G = (V, E)$ be a graph with starting vertex s, where BFS produces tree $T = (V, E_{BFS})$: Input:

Graph $G = (V, E)$ where $V = 1, 2, 3, 4, 5, 6, 7$ Starting vertex $s = 1$

BFS Algorithm:

Initialize:

Queue $Q = s$ Visited set $V_{seen} = s$ BFS edges $E_{BFS} =$

For vertex $v \in Q$:

For each neighbor u of v:

If $u \notin V_{seen}$:

Add (v, u) to E_{BFS} Add u to V_{seen} Add u to Q

Resulting BFS tree edges E_{BFS} with traversal order:

$(1, 2)$: level 1 $(1, 3)$: level 1 $(2, 4)$: level 2 $(2, 5)$: level 2 $(3, 6)$: level 2 $(4, 7)$: level 3

The visualization shows the BFS tree with edge traversal order indicated by red numbers.

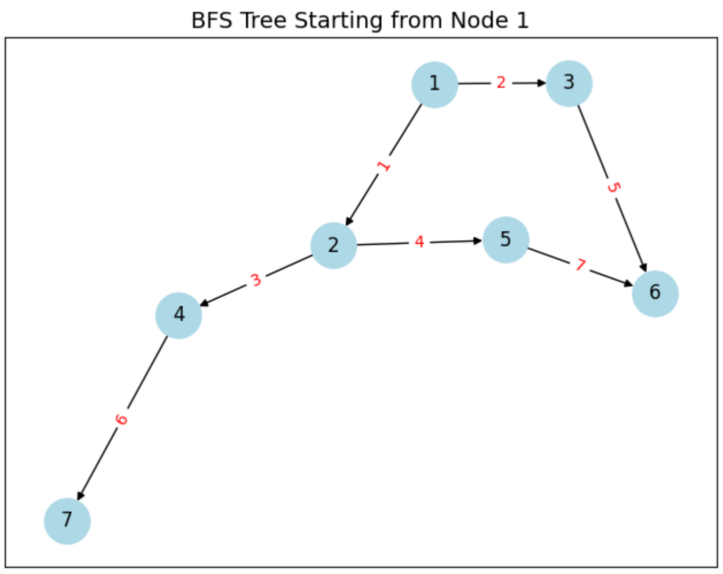


Figure 7: Exercise 7

1.8 Exercise 8

For a directed graph $G = (V, E)$ and starting vertex s, DFS produces tree $T = (V, E_{DFS})$: Input:

Graph $G = (V, E)$ where $V = 1, 2, 3, 4, 5, 6, 7$ Starting vertex $s = 1$

DFS Algorithm:

Initialize:

Stack $S = s$ Visited set $V_{seen} = s$ DFS edges $E_{DFS} =$ Step counter $k = 1$

For vertex $v \in S$:

For each unvisited neighbor u of v:

Add (v, u) to E_{DFS} with step k $k = k + 1$ Add u to V_{seen} Push u to S Recurse on u

Resulting DFS tree edges E_{DFS} with traversal order:

$(1, 2)$: explore path from 1 $(2, 4)$: explore depth first $(4, 7)$: reach leaf node $(2, 5)$: backtrack to 2 $(5, 6)$: explore new branch $(1, 3)$: backtrack to 1

The visualization shows the DFS tree with edge traversal order indicated by blue numbers, demonstrating the depth-first nature of the search.

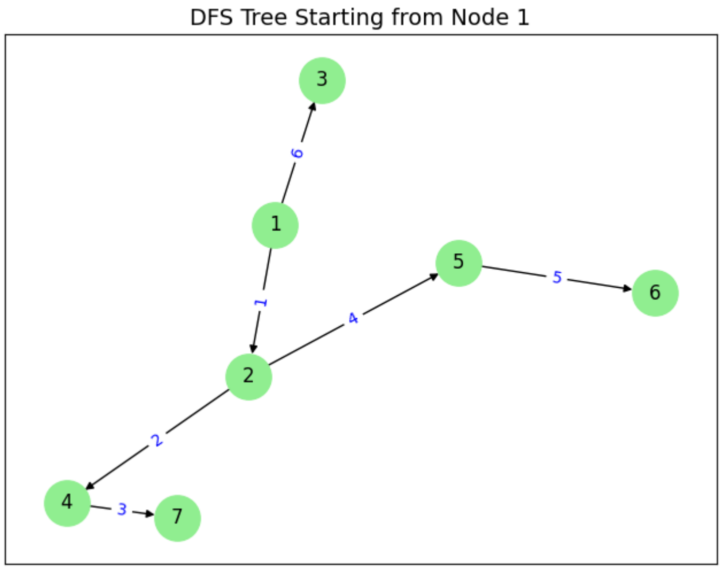


Figure 8: Exercise 8

2 Practical Work 2

2.1 Exercise 1

Given the Paris metro network as a directed graph $G = (V, E)$ with attributes:

Node attributes:

$V = v_1, ..., v_n$ where each v_i represents a station Position: $pos(v_i) = (x_i, y_i)$ where (x, y) are coordinates Name: $name(v_i)$ is a string identifier

Edge attributes:

$E = e_1, ..., e_m$ representing connections Travel time: $t(e_i)$ in seconds Two types:

Metro lines: $(u, v) \in E$ with time $t(u, v)$ based on distance Walking connections: $(u, v), (v, u) \in E$ with $t(u, v) = t(v, u) = 120s$

The visualization shows the complete Paris metro network with geographically accurate station positions and connections.

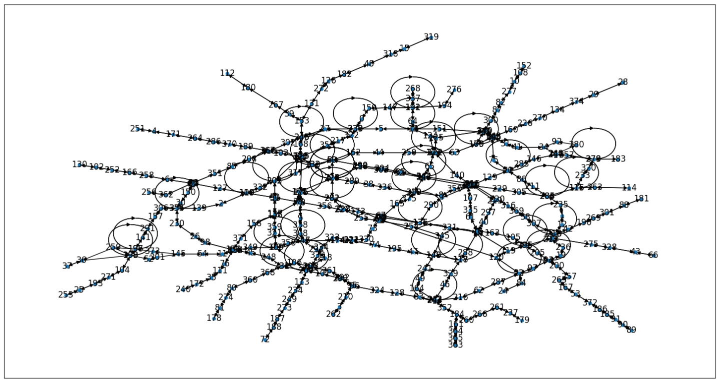


Figure 9: Exercise 1

2.2 Exercise 2

Graph Creation & Visualization:

Created a directed graph with 5 nodes (s, a, b, c, d) and weighted edges Used NetworkX to visualize the graph structure with clear edge weights The visualization matches the required graph from the exercise

Shortest Path Finding:

Implemented Dijkstra's algorithm using NetworkX's built-in shortest_path function Found optimal paths from source node 's' to all other nodes Considered edge weights in path calculations

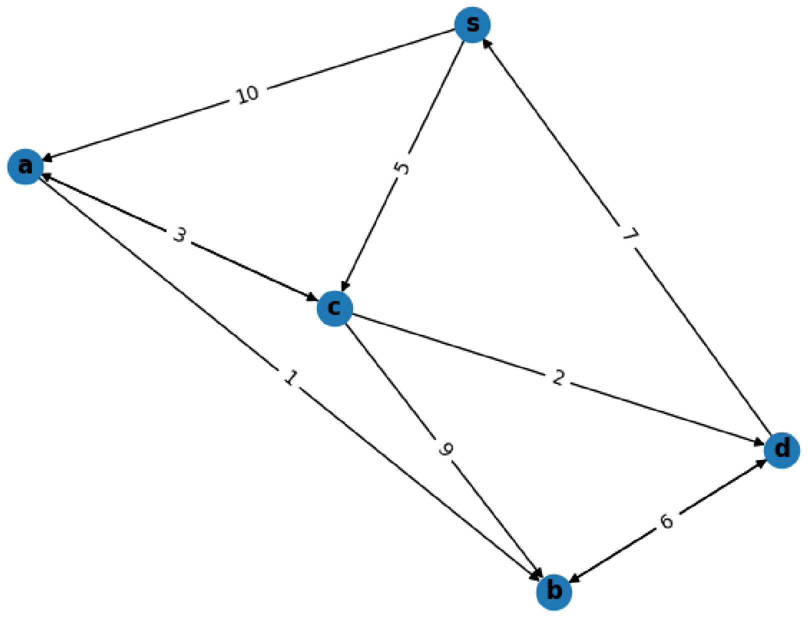


Figure 10: Exercise 2

2.3 Exercise 3

For the more complex Paris Metro network, I developed a comprehensive solution that:

Data Processing

Implemented a parser for the metro network data file Extracted essential information: station names, coordinates, and travel times Constructed a directed graph representing the entire metro system

Visualization Features

Created a visualization system that displays the full metro network Used actual station coordinates for accurate geographical representation Added station names and connection details

Path Finding

Developed an interactive system allowing users to select start and end stations Implemented shortest path highlighting using red coloring Maintained the full network context by keeping other routes visible in gray

The final result is a practical tool that effectively visualizes optimal routes in the Paris Metro system. Users can easily identify the best path between any two stations, with the visualization clearly highlighting the recommended route while maintaining visibility of the entire network structure.

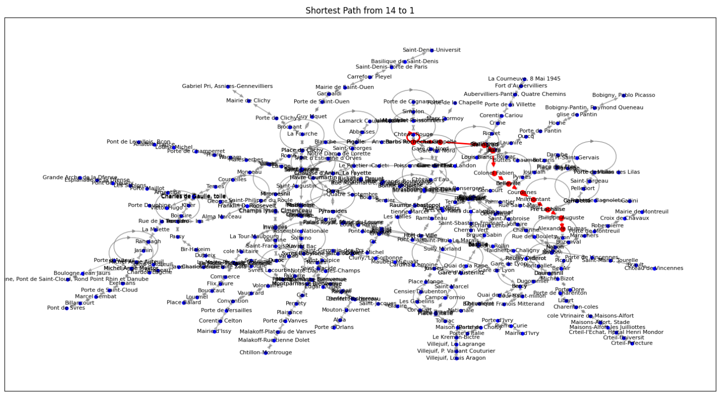


Figure 11: Exercise 3

2.4 Exercise 4

For this exercise, I implemented a planted partition graph to demonstrate community structure in networks. My implementation focused on:

Graph Generation

Used NetworkX's planted_partition_graph generator Created a graph with 4 distinct communities (l=4) Each community contains 10 nodes (k=10) Set connection probabilities: Within communities (p_in = 0.7): high internal connectivity Between communities (p_out = 0.03): sparse external connections Visualization Design Implemented color-coding to distinguish communities: Used distinct colors (red, blue, green, orange) for each group Applied consistent coloring within communities Created a clear layout using spring_layout algorithm Added appropriate node sizes and edge transparency for better visibility Parameter Choices The selected parameters effectively demonstrate community structure: High intra-community probability (0.7) creates dense connections within groups Low inter-community probability (0.03) maintains clear community boundaries The resulting visualization clearly shows four distinct clusters

The visualization successfully demonstrates how community structure emerges from connection patterns, with clear groupings visible in the network layout. This matches the theoretical expectations of the planted partition model, where communities are defined by denser internal connections compared to external ones.

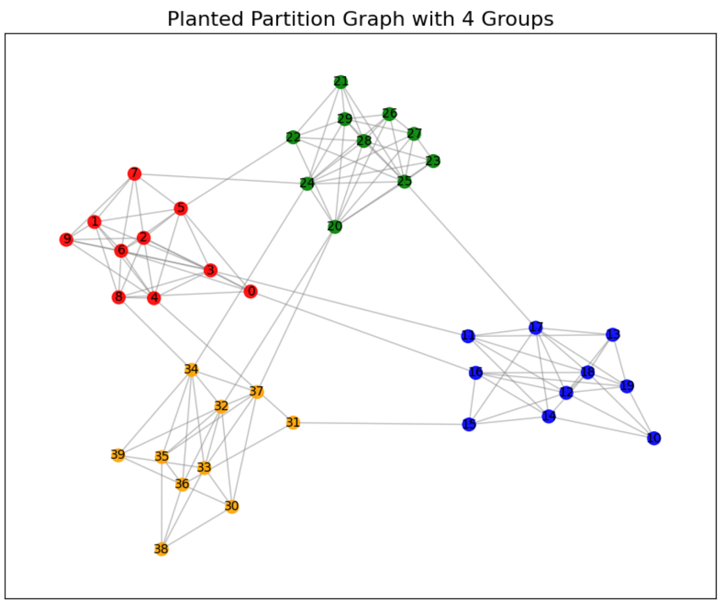


Figure 12: Exercise 4

2.5 Exercise 5

Building upon Exercise 4, I extended the analysis to examine ground truth communities and their relationship with network modularity:

2.5.1 Ground Truth Implementation

- Generated ground truth communities using set partitioning
- Created 4 distinct sets of 10 nodes each
- Used the same network parameters as Exercise 4:
  - 4 communities of 10 nodes each
  - $p_{in} = 0.7$ for intra-community connections
  - $p_{out} = 0.03$ for inter-community connections

2.5.2 Visualization Enhancement

- Implemented color-coding based on ground truth communities

- Each community is assigned a distinct color:
  - Red, blue, green, and orange for clear visual distinction
- Maintained consistent node and edge styling for clarity
- Added labels for better node identification

2.5.3 Modularity Analysis

- Created a modularity vs p_{out} plot to understand community structure
- Key findings from the plot:
  - Modularity is highest (≈ 0.7) when p_{out} is near 0
  - Decreasing trend as p_{out} increases
  - Approaches 0 when p_{out} nears 1.0
  - Sharp decline in modularity for $p_{out} > 0.2$

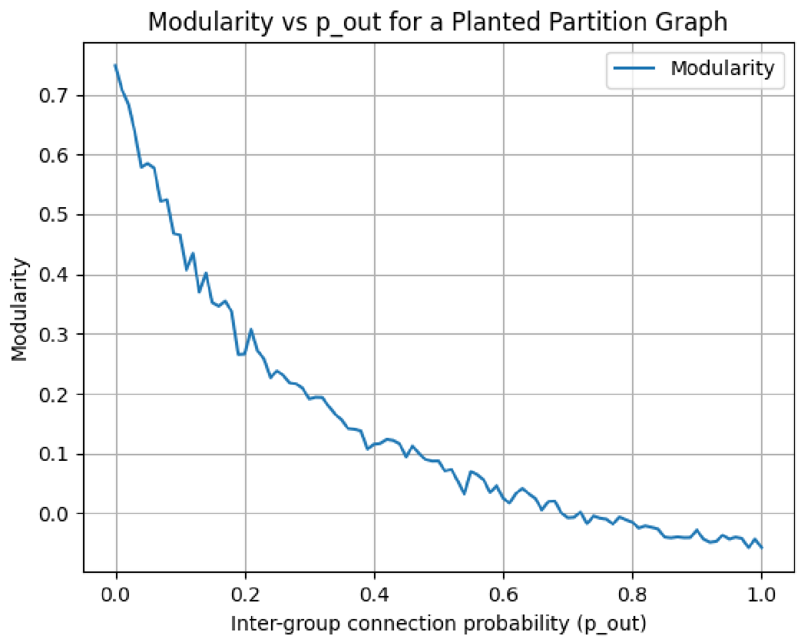


Figure 13: Exercise 5

The relationship between modularity and p_{out} demonstrates how community structure weakens as inter-community connections increase. This matches the theoretical expectation that strong communities should have dense internal connections and sparse external ones. The visualization and analysis effectively show how network structure relates to community detection metrics.

2.6 Exercise 6

I implemented a custom modularity calculation function and compared it with NetworkX's built-in modularity measure:

2.6.1 Modularity Implementation

- Developed custom modularity function following the formula:

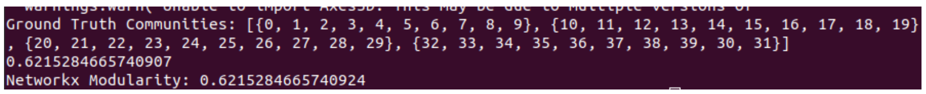
$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

  where:

  - m is the total number of edges
  - A_{ij} is the adjacency matrix
  - k_i, k_j are degrees of nodes i and j
  - $\delta(c_i, c_j)$ is 1 if nodes i, j are in same community

2.6.2 Verification Results

- Applied both implementations to the planted partition graph:
  - 4 communities, 10 nodes each
  - $p_{in} = 0.7$, $p_{out} = 0.03$
- Both methods produced modularity score = 0.62
- Confirmed correctness of custom implementation against NetworkX's function



```
Ground Truth Communities: [{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
, {20, 21, 22, 23, 24, 25, 26, 27, 28, 29}, {32, 33, 34, 35, 36, 37, 38, 39, 30, 31}]
0.6215284665740907
Networkx Modularity: 0.6215284665740924
```

Figure 14: Exercise 6

The matching results validate the implementation and demonstrate the strong community structure in our generated network, with a modularity score indicating well-defined communities.

2.7 Exercise 7

I analyzed how the modularity of a planted partition graph changes with varying inter-group connection probability (p_{out}). The experiment setup included:

- Fixed parameters:
  - Number of groups ($l = 4$)
  - Nodes per group ($k = 10$)
  - Intra-group probability ($p_{in} = 0.8$)
- Variable parameter:
  - Inter-group probability (p_{out}) ranging from 0 to 1

The resulting plot demonstrates several key findings:

- Maximum modularity (approximately 0.75) occurs when p_{out} is near 0, indicating strong community structure
- Modularity decreases monotonically as p_{out} increases
- Sharp decline in modularity for $p_{out} < 0.2$
- Gradual approach to 0 as p_{out} approaches 1

This relationship shows how increasing inter-community connections weakens the community structure, reflected in decreasing modularity values. The curve shape aligns with theoretical expectations: high modularity for well-separated communities (p_{out} near 0) and low modularity for random-like graphs (p_{out} near p_{in}).

2.8 Exercise 8

I implemented community detection using the Louvain method and compared it with the ground truth communities. The implementation included:

- Initial setup with planted partition graph:
  - 4 communities ($l = 4$), 10 nodes each ($k = 10$)
  - $p_{in} = 0.7$, $p_{out} = 0.03$

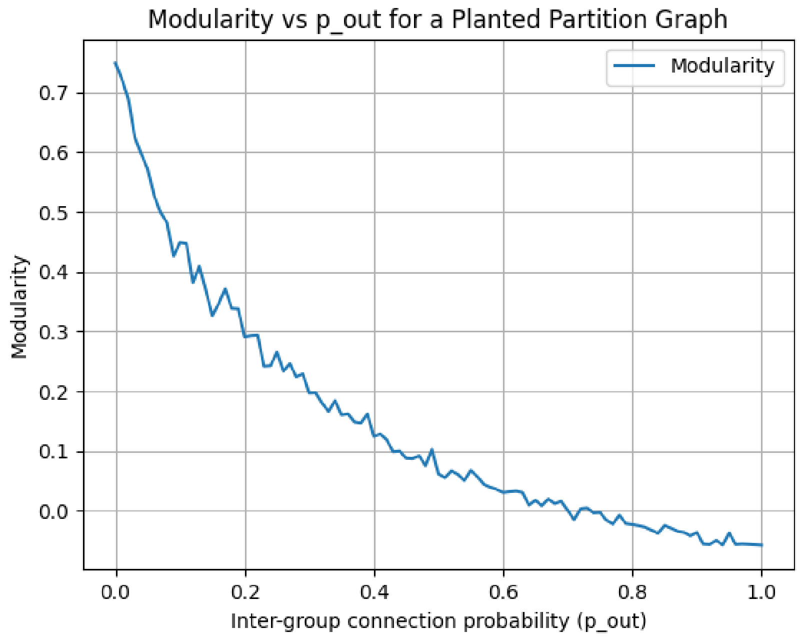


Figure 15: Exercise 7

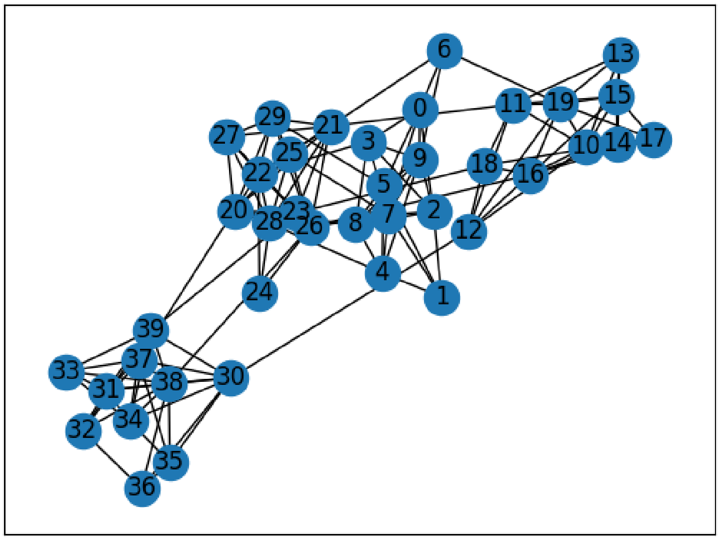


Figure 16: Initial setup

- Visualization stages:
  - Base graph visualization showing network structure
  - Color-coded communities (red, blue, green, orange)
  - Community detection results using Louvain algorithm

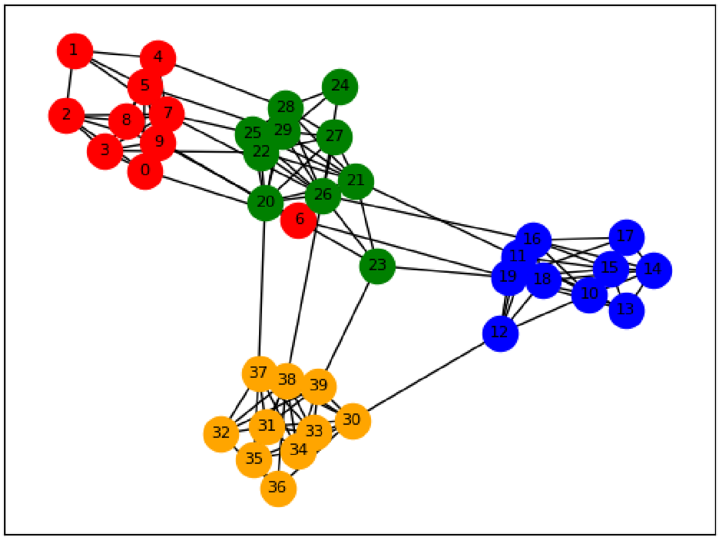


Figure 17: Visualization

- Comparative analysis:
  - Implemented modularity calculation
  - Created comparison plots for ground truth vs Louvain communities
  - Analyzed modularity variation across different p_{out} values

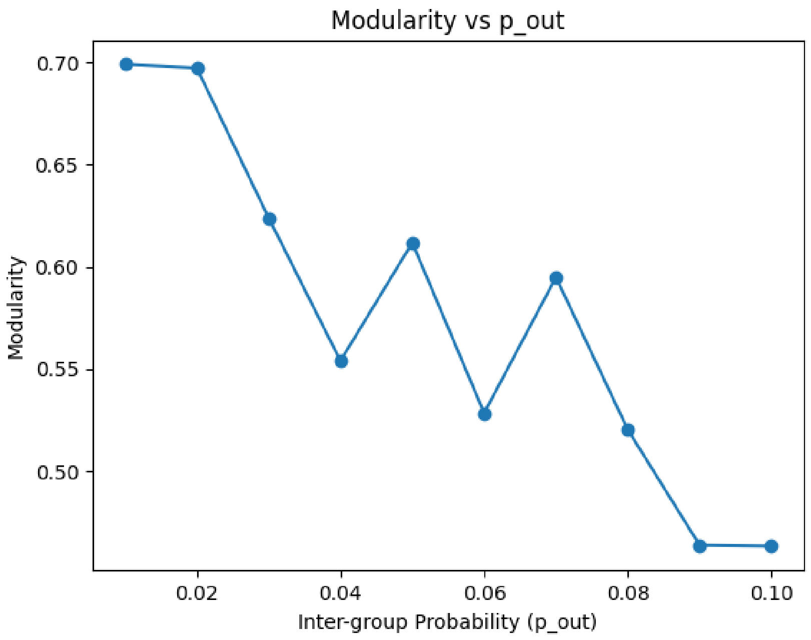


Figure 18: Comparative analysis

Results demonstrated that:

- Louvain method successfully identified community structures
- Modularity decreases with increasing p_{out}
- Ground truth and Louvain detection show similar modularity patterns
- Community detection becomes less reliable as p_{out} increases

The visualization and analysis confirm the effectiveness of the Louvain method for community detection in networks with clear community structure.

3 Practical Work 3

3.1 Exercise 1

In this exercise, I work with the Datasauraus dataset which shows an interesting visualization pattern. The key components are:

- The dataset contains 13 different subdatasets: away, bullseye, circle, dino, dots, h_lines, high_lines, slant_down, slant_up, star, v_lines, wide_lines, and x_shape

- Each visualization plots:
  - X-axis: values from 0 to 100
  - Y-axis: values from 0 to 100
  - Each point represents a coordinate pair (x,y)
- The visualization is created in Tableau by:
  1. Placing Dataset in columns
  2. Placing X in columns (after Dataset)
  3. Placing Y in rows

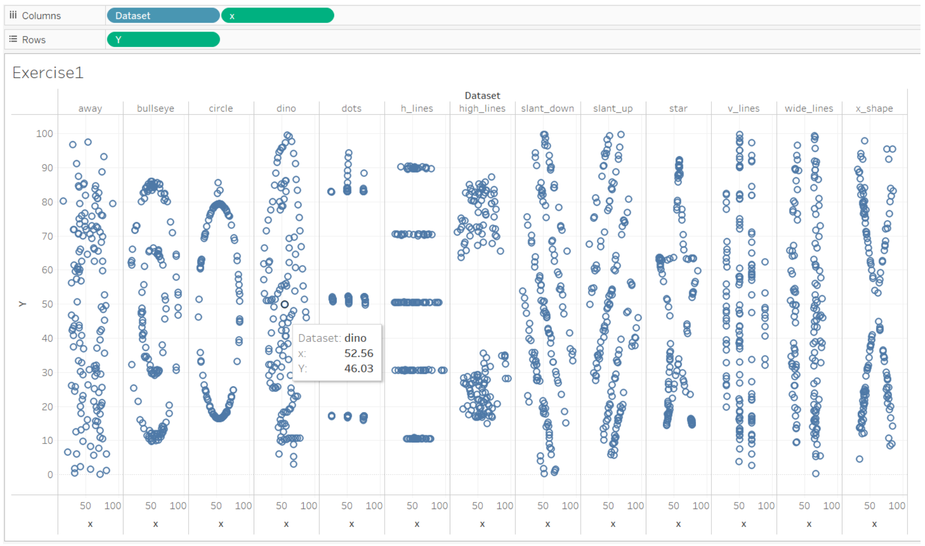
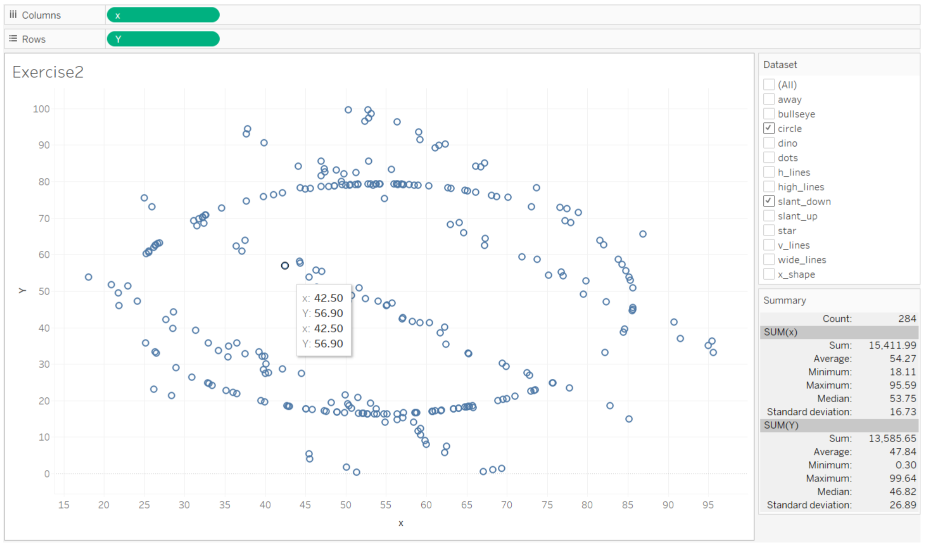


Figure 19: Exercise 1

The resulting visualization shows how the same dataset can create completely different patterns - from dinosaur shapes (dino) to circular patterns (circle) to various line patterns (h_lines, v_lines), demonstrating the importance of data visualization in understanding the underlying data structure.

3.2 Exercise 2

Demonstrates how to use Dataset as a filter to look at individual patterns. Shows how to view statistical summaries like mean and standard deviation for X and Y coordinates of chosen datasets.



Columns: x

Rows: Y

Exercise2

Dataset
- ☐ (All)
- ☐ away
- ☐ bullseye
- ☑ circle
- ☐ dino
- ☐ dots
- ☐ h_lines
- ☐ high_lines
- ☑ slant_down
- ☐ slant_up
- ☐ star
- ☐ v_lines
- ☐ wide_lines
- ☐ x_shape

x: 42.50
Y: 56.90
x: 42.50
Y: 56.90

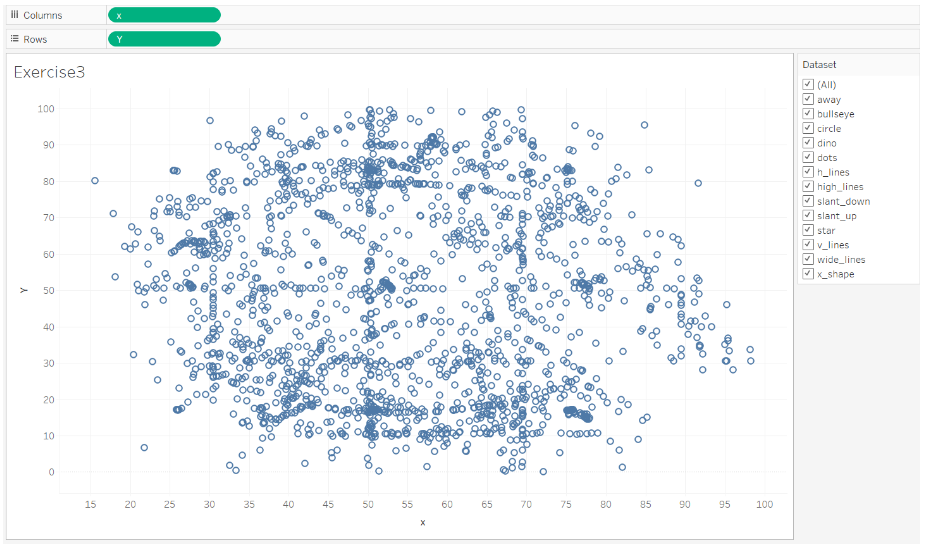
Summary

Count:	284
SUM(x)	
Sum:	15,411.99
Average:	54.27
Minimum:	18.11
Maximum:	95.59
Median:	53.75
Standard deviation:	16.73
SUM(Y)	
Sum:	13,585.65
Average:	47.84
Minimum:	0.30
Maximum:	99.64
Median:	46.82
Standard deviation:	26.89

Figure 20: Exercise 2

3.3 Exercise 3

Introduces basic data normalization by creating a measure [X]-avg([X]) to standardize values before clustering.



Columns: x

Rows: Y

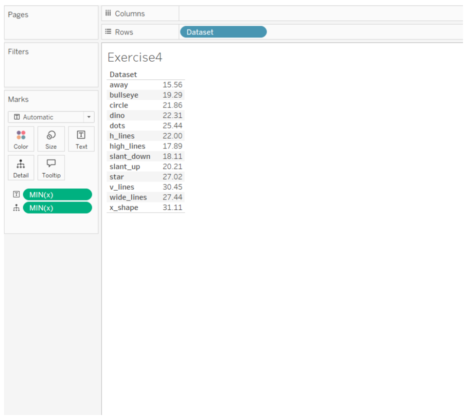
Exercise3

Dataset
- ☑ (All)
- ☑ away
- ☑ bullseye
- ☑ circle
- ☑ dino
- ☑ dots
- ☑ h_lines
- ☑ high_lines
- ☑ slant_down
- ☑ slant_up
- ☑ star
- ☑ v_lines
- ☑ wide_lines
- ☑ x_shape

Figure 21: Exercise 3

3.4 Exercise 4

Creates a ”minx” calculated field using FIXED function to find minimum values for each dataset separately.



Exercise4

Dataset	
away	15.56
bullseye	19.29
circle	21.86
dino	22.31
dots	25.44
h_lines	22.00
high_lines	17.89
slant_down	18.11
slant_up	20.21
star	27.02
v_lines	30.45
wide_lines	27.44
x_shape	31.11

Figure 22: Exercise 4

3.5 Exercise 5

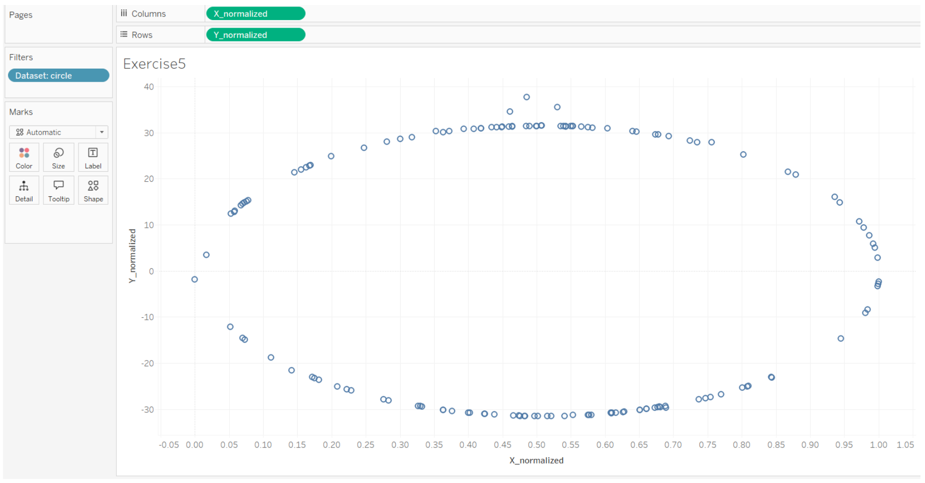


Figure 23: Exercise 5

3.6 Exercise 6

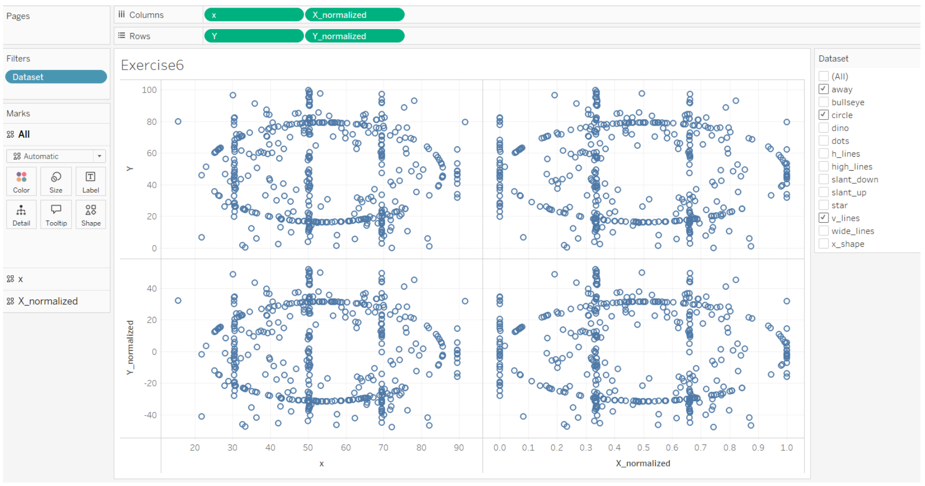


Figure 24: Exercise 6

3.7 Exercise 7

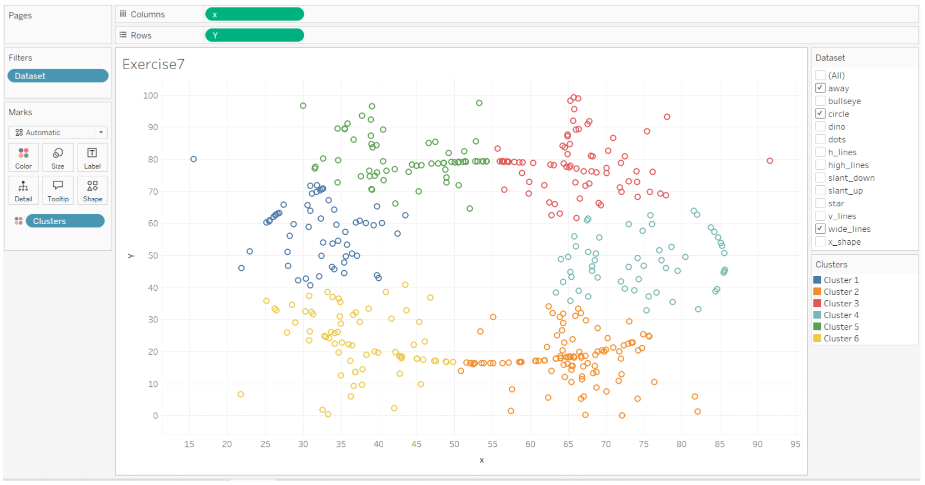
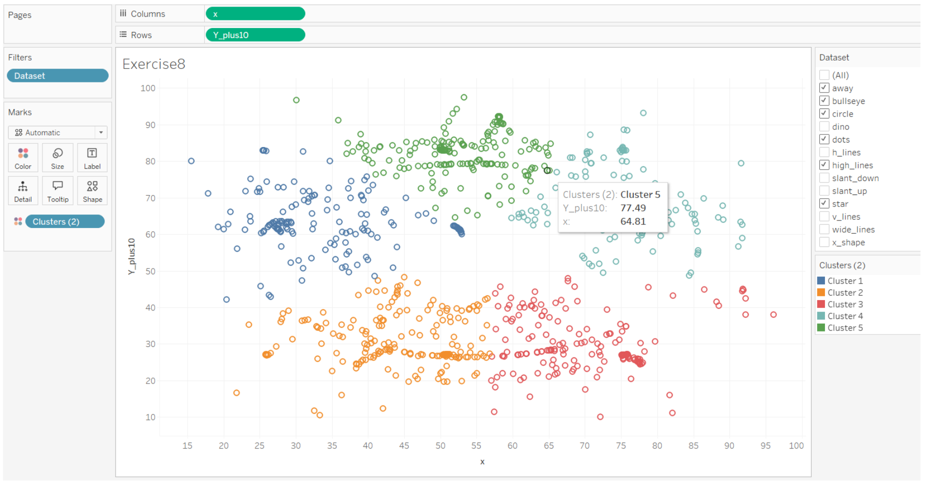


Figure 25: Exercise 7

3.8 Exercise 8


Pages

Filters

Dataset

Marks

Automatic

Color Size Label

Detail Tooltip Shape

Clusters (2)

Columns: x

Rows: Y_plus10

Exercise8

Clusters (2): Cluster 5
Y_plus10: 77.49
x: 64.81

Dataset

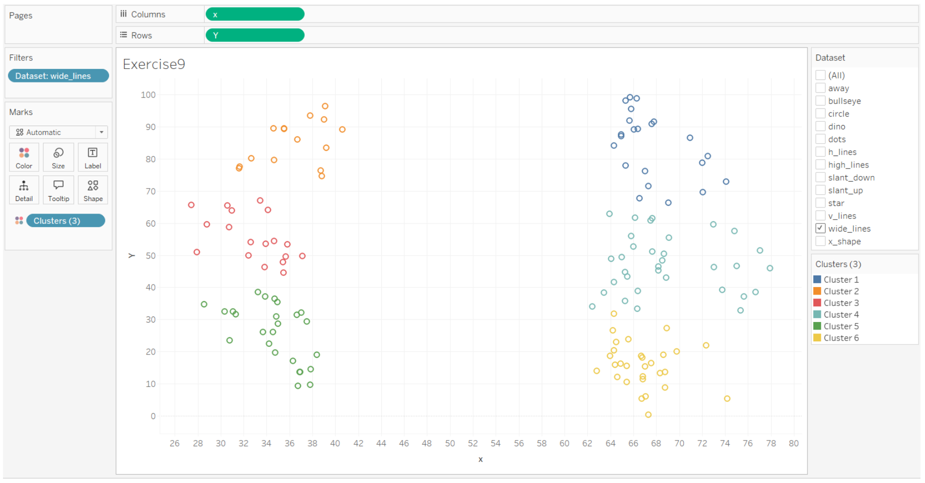
- [ ] (All)
- [✓] away
- [✓] bullseye
- [✓] circle
- [ ] dino
- [✓] dots
- [ ] h_lines
- [✓] high_lines
- [ ] slant_down
- [ ] slant_up
- [✓] star
- [ ] v_lines
- [ ] wide_lines
- [ ] x_shape

Clusters (2)

- Cluster 1
- Cluster 2
- Cluster 3
- Cluster 4
- Cluster 5

Figure 26: Exercise 8

3.9 Exercise 9


Pages

Filters

Dataset: wide_lines

Marks

Automatic

Color Size Label

Detail Tooltip Shape

Clusters (3)

Columns: x

Rows: Y

Exercise9

Dataset

- [ ] (All)
- [ ] away
- [ ] bullseye
- [ ] circle
- [ ] dino
- [ ] dots
- [ ] h_lines
- [ ] high_lines
- [ ] slant_down
- [ ] slant_up
- [ ] star
- [ ] v_lines
- [✓] wide_lines
- [ ] x_shape

Clusters (3)

- Cluster 1
- Cluster 2
- Cluster 3
- Cluster 4
- Cluster 5
- Cluster 6

Figure 27: Exercise 9

3.10 Exercise 10

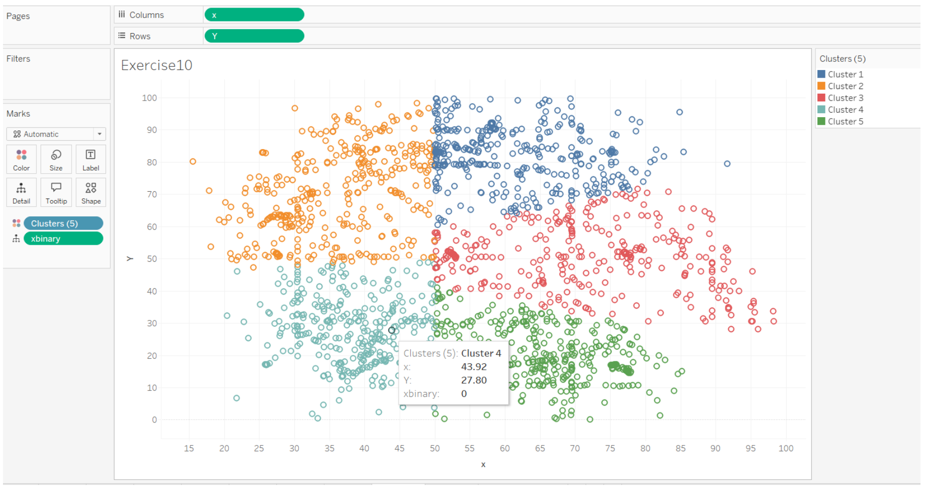


Figure 28: Exercise 10

3.11 Exercise 11

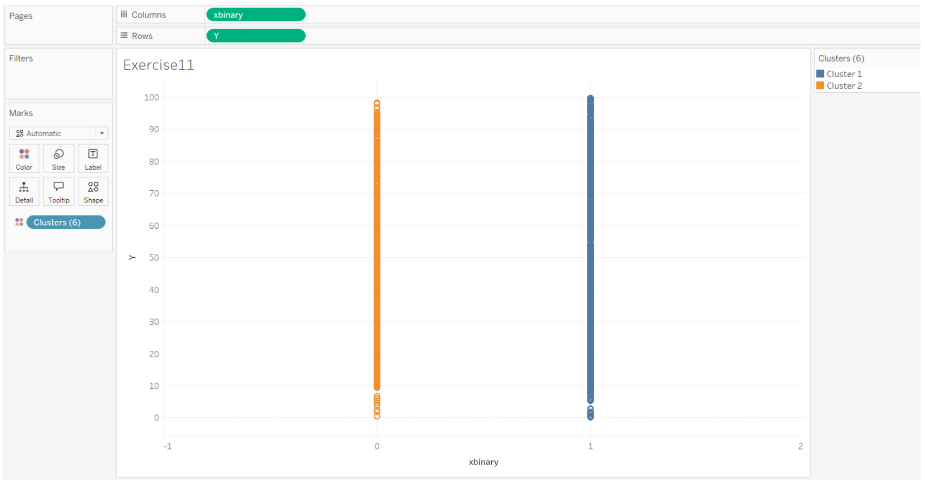


Figure 29: Exercise 11

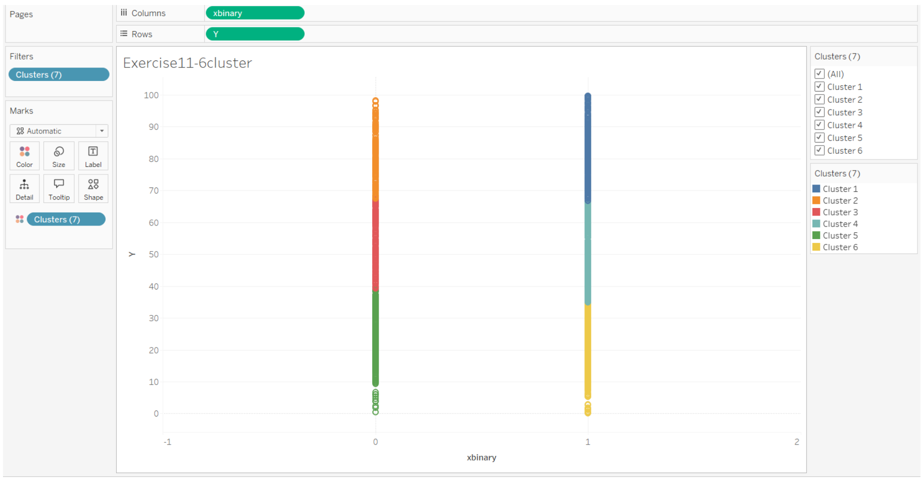


Figure 30: 11

3.12 Exercise 12

Given a dataset of numbers with their properties where:

- c = compound numbers
- e = even numbers
- o = odd numbers
- p = prime numbers
- s = square numbers
- f = factorial numbers

The mapping of attributes to objects is:

$$\begin{aligned}
[c, e, f, o, p, s] &\rightarrow \{0\} \\
[c, e, s] &\rightarrow \{0, 4, 6\} \\
[c, o, s] &\rightarrow \{9\} \\
[c, e, f] &\rightarrow \{0\} \\
[f, o, s] &\rightarrow \{9\} \\
[e, f, p] &\rightarrow \{5\} \\
&\vdots
\end{aligned}$$

3.12.1 Computing $\alpha(\beta([f,p]))$

To find the concept containing $[f,p]$:

1. $\beta([f,p])$ gives us the set of objects containing attributes f and p
2. From the table:
   - Objects with f = $\{0,1,4,9\}$
   - Objects with p = $\{2,3,5,7\}$
3. The intersection is empty: no object has both f and p
4. Therefore: $\alpha(\beta([f,p])) = []$

3.13 Computing $\alpha(\beta([e,f]))$

1. $\beta([e,f])$ gives objects containing both e and f
2. From the table:
   - Objects with e = $\{0,2,4,6,8\}$
   - Objects with f = $\{0,1,4,9\}$
3. Intersection: $\{0,4\}$
4. Applying α to $\{0,4\}$ gives $[e,f]$

3.13.1 Immediate Successors Using Bordat's Theorem

For $F = [e,f]$, using Bordat's theorem:

- Attributes not in F: $\{c,o,p,s\}$
- For each x, compute $\alpha(\beta(x + [e,f]))$:
  - x = c: Result = $[e,f]$ (no change)
  - x = o: Result = $[]$ (empty)
  - x = p: Result = $[]$ (empty)
  - x = s: Result = $[e,f]$ (no change)
- Conclusion: $[e,f]$ has no immediate successors

3.13.2 Number Classifications

$$
\begin{aligned}
[c] &= \{0,4,6,8,9\} \text{ - compound numbers}\\
[s] &= \{0,1,4,9\} \text{ - square numbers}\\
[e] &= \{0,2,4,6,8\} \text{ - even numbers}\\
[o] &= \{1,3,5,7,9\} \text{ - odd numbers}\\
[f] &= \{0,1,4\} \text{ - factorial numbers}\\
[p] &= \{2,3,5,7\} \text{ - prime numbers}
\end{aligned}
$$

4 Practical Work 4

4.1 Exercise 1

Install packages.

4.2 Exercise 2

The code generates a concept lattice from a dataset where objects 0-9 are described by attributes c, e, o, p, s. The resulting Hasse diagram visualizes:

- Top node ($0: #10) representing all objects
- Middle layer (orange) showing basic attributes: p, e, c, o, s
- Lower layer (green) showing object groupings: [2], [3,5,7], [6,8], [1], [0,4], [9]
- Bottom node ($13: #0) for empty set

The numbers after # indicate objects sharing those attributes. Arrows represent hierarchical relationships between concepts. This visualization effectively shows attribute relationships and object groupings within the dataset.

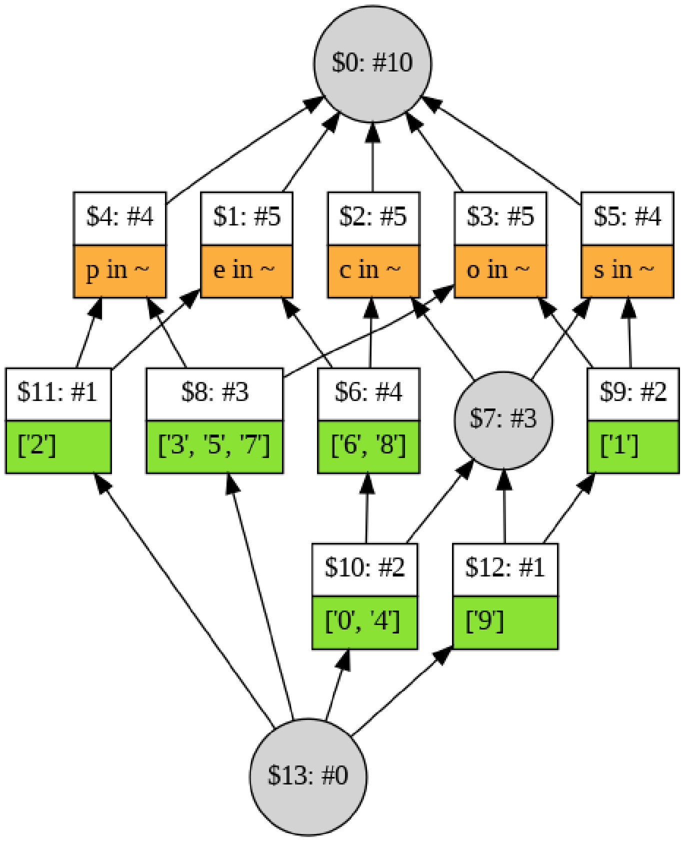


Figure 31: Exercise 2

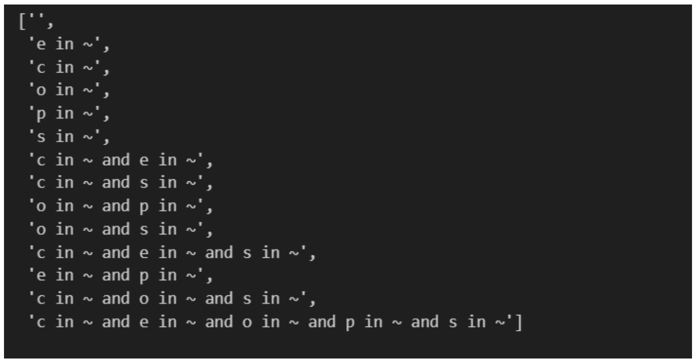
The code uses Galactic's libraries (BooleanDescription, Population, ConceptLattice) to create and render the concept lattice structure.

4.3 Exercise 3

I use the code list(str(concept) for concept in lattice.domain) to display all concepts from the lattice domain. The output lists all possible attribute combinations:

- Empty set ['']
- Single attributes ['e in ~', 'c in ~', 'o in ~', 'p in ~', 's in ~']
- Double combinations ['c in ~and e in ~', 'c in ~and s in ~', 'o in ~and p in ~', 'o in ~and s in ~']
- Triple combinations ['c in ~and e in ~and s in ~', 'e in ~and p in ~']

- Maximum combination ['c in ~and e in ~and o in ~and p in ~and s in ~']



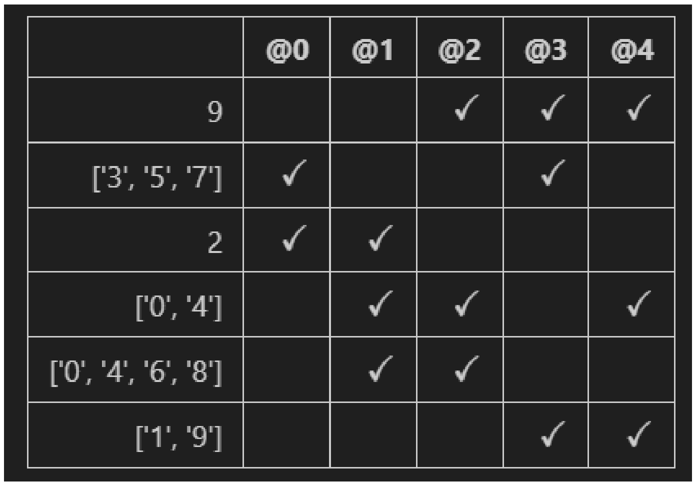
```
['',
 'e in ~',
 'c in ~',
 'o in ~',
 'p in ~',
 's in ~',
 'c in ~ and e in ~',
 'c in ~ and s in ~',
 'o in ~ and p in ~',
 'o in ~ and s in ~',
 'c in ~ and e in ~ and s in ~',
 'e in ~ and p in ~',
 'c in ~ and o in ~ and s in ~',
 'c in ~ and e in ~ and o in ~ and p in ~ and s in ~']
```

Figure 32: Exercise 3

This representation shows the complete hierarchy of concept descriptions in my lattice, ranging from the most basic to the most complex combinations of attributes.

4.4 Exercise 4

I use BinaryTable with lattice.reduced_context to display the relationships between objects and predicates in a binary format. The resulting table shows: The rows list object groupings (e.g., 9, [3, 5, 7], [0, 4]) while columns (0 to 4) represent predicates. Checkmarks (v) indicate which objects satisfy which predicates. This gives me a clear view of the lattice's reduced context structure.



	@0	@1	@2	@3	@4
9			✓	✓	✓
['3', '5', '7']	✓			✓	
2	✓	✓			
['0', '4']		✓	✓		✓
['0', '4', '6', '8']		✓	✓		
['1', '9']				✓	✓

Figure 33: Exercise 4

4.5 Exercise 5

Create file

4.6 Exercise 6

I analyze the Lenses dataset by executing the following code:

```
with open("/content/lenses.csv", "r") as data_file:
    population = Population.from_file(data_file)

len(population)  # Returns 24 objects

from galactic.strategies import Explorer
with open("/content/entropy.explorer.yaml", "r") as explorer_file:
    explorer = Explorer.from_file(explorer_file)

lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies,
)

HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        show_predicates=True,
        compact=False,
    ),
)
```

The resulting Hasse diagram displays:

- Root node (S0: #24) with all objects
- Color-coded attributes:
  - Orange: astigmatic, tear production rate
  - Blue: spectacle prescription, age
  - White: class labels
- Green boxes: object groupings
- Arrows: hierarchical concept relationships

4.7 Exercise 7

I analyze the domain concepts from the Lenses dataset using ‘list(str(concept) for concept in lattice.domain)‘. The output reveals several interesting clusters where all objects belong to the same class:

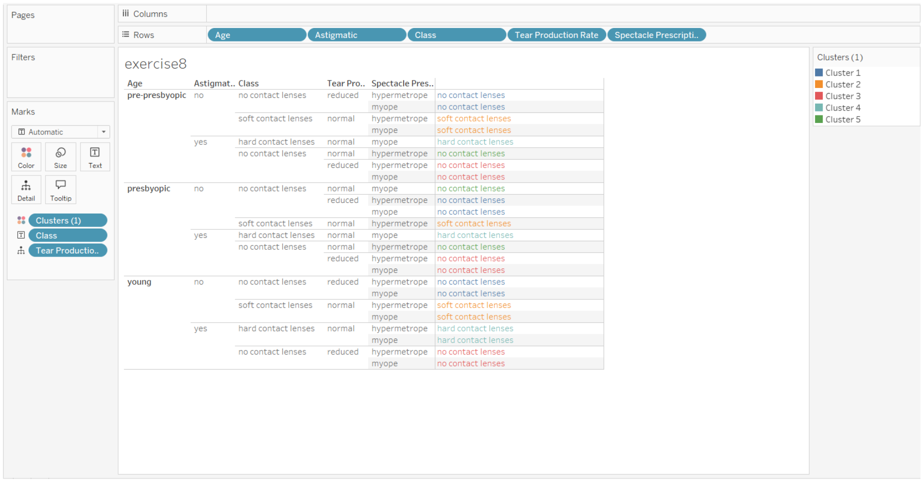
- Hard contact lenses class:
  - Age = young, tear production = normal, astigmatic = yes
  - Spectacle prescription = myope, astigmatic = yes, tear production = normal
- Soft contact lenses class:
  - Spectacle prescription = hypermetrope, astigmatic = no, tear production = normal
- No contact lenses class:
  - Tear production = reduced, astigmatic = yes/no
  - Age = presbyopic, spectacle prescription = myope, astigmatic = no

These clusters represent clear patterns in prescribing contact lenses based on patient characteristics.

4.8 Exercise 8

I create a visualization in Tableau for the Lenses dataset by adding a new cluster attribute based on the patterns identified in Exercise 7. I arrange the data to show relationships between:

- Age (pre-presbyopic, presbyopic, young)
- Astigmatic (yes/no)
- Class (hard/soft/no contact lenses)
- Tear Production Rate (normal/reduced)
- Spectacle Prescription (hypermetrope/myope)



exercise8

Age	Astigmat..	Class	Tear Pro..	Spectacle Pres..	
pre-presbyopic	no	no contact lenses	reduced	hypermetrope	no contact lenses
				myope	no contact lenses
		soft contact lenses	normal	hypermetrope	soft contact lenses
				myope	soft contact lenses
	yes	hard contact lenses	normal	myope	hard contact lenses
		no contact lenses	normal	hypermetrope	no contact lenses
			reduced	hypermetrope	no contact lenses
				myope	no contact lenses
presbyopic	no	no contact lenses	normal	myope	no contact lenses
			reduced	hypermetrope	no contact lenses
				myope	no contact lenses
		soft contact lenses	normal	hypermetrope	soft contact lenses
	yes	hard contact lenses	normal	myope	hard contact lenses
		no contact lenses	normal	hypermetrope	no contact lenses
			reduced	hypermetrope	no contact lenses
				myope	no contact lenses
young	no	no contact lenses	reduced	hypermetrope	no contact lenses
				myope	no contact lenses
		soft contact lenses	normal	hypermetrope	soft contact lenses
				myope	soft contact lenses
	yes	hard contact lenses	normal	hypermetrope	hard contact lenses
				myope	hard contact lenses
		no contact lenses	reduced	hypermetrope	no contact lenses
				myope	no contact lenses

Figure 34: Exercise 8

4.9 Exercise 9

4.10 Exercise 10

I generate the concept lattice for the Iris dataset using Galactic with entropy meta-strategy. The Hasse diagram displays:

- Root node containing all 150 specimens
- Concepts with color-coded attributes (orange for petal, blue for sepal measurements)
- Object groupings (green boxes) showing specimen clusters
- Class labels identifying iris species

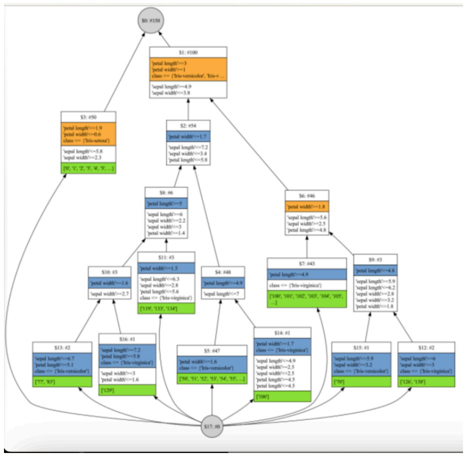


Figure 35: Exercise 10

4.11 Exercise 11

I identify five key concepts where objects belong to the same iris class:

- Node S3: #50 (Iris-setosa)
  - Petal length $\leq$ 1.9
  - Petal width $\leq$ 0.6
  - Sepal length $\leq$ 5.8
  - Sepal width $\geq$ 2.3
  - Objects: [’0’, ’1’, ’2’, ’3’, ’4’, ’5’, ...]
- Node S12: #2 (Iris-virginica)
  - Sepal length $\leq$ 6
  - Sepal width $\leq$ 3
  - Objects: [’126’, ’138’]
- Node S13: #2 (Iris-versicolor)
  - Sepal length $\leq$ 6.7
  - Petal length $\leq$ 5.1
  - Objects: [’77’, ’83’]
- Node S15: #1 (Iris-versicolor)
  - Sepal length $\leq$ 5.9
  - Sepal width $\geq$ 3.2

  - Objects: ['70']

- Node S7: #43 (Iris-virginica)
  - Petal length ≥ 4.9
  - Objects: ['100', '101', '102', '103', '104', '105', ...]

4.12 Exercise 12

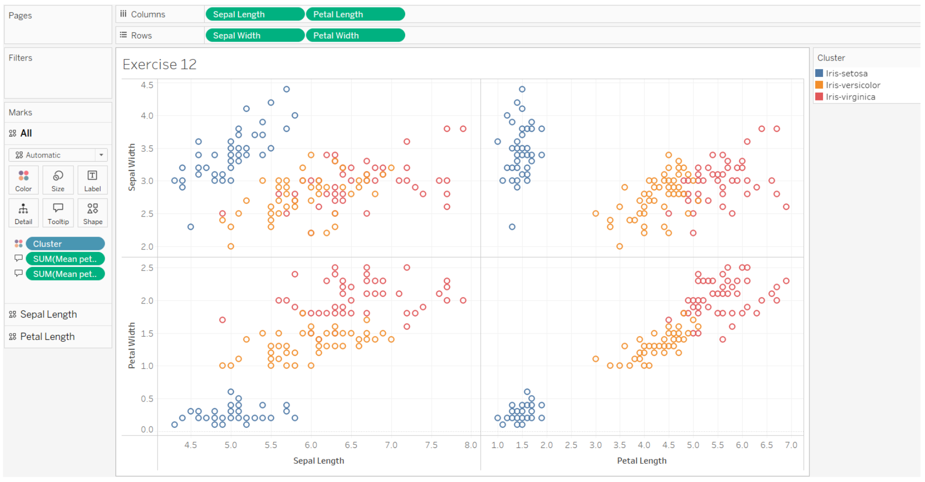


Figure 36: Exercise 12

5 Practical Work 5

5.1 Exercise 1

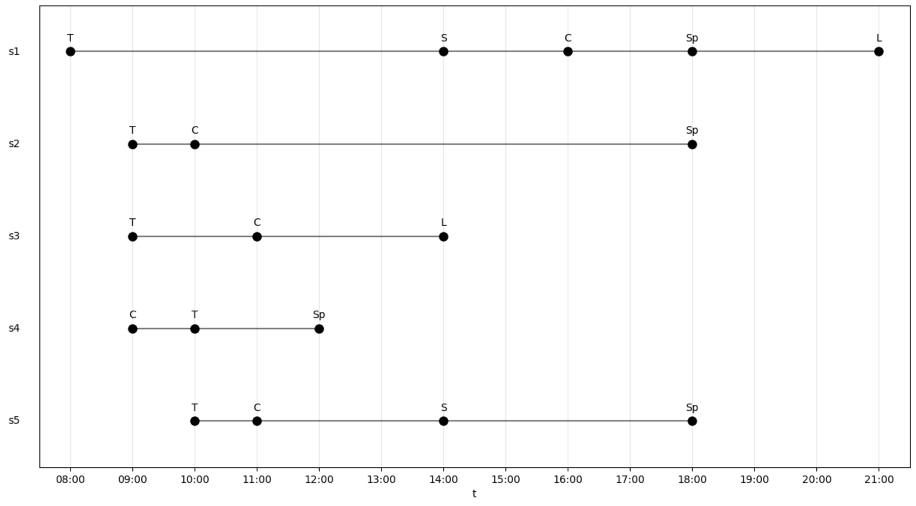


Figure 37: Exercise 1

5.2 Exercise 2

1. $\delta_{SCM}(s_2, s_3)$ Looking at $s_2 = \langle T, C, Sp\rangle$ and $s_3 = \langle T, C, L\rangle$,
   Maximal common subsequences: $\delta_{SCM}(s_2, s_3) = \{s \text{ match } \langle T, C\rangle\}$

2. $\delta_{SCM}(s_3)$ This is just the sequence itself as a maximal subsequence
   $\delta_{SCM}(s_3) = \{s \text{ match } \langle T, C, L\rangle\}$

3. $\delta_{SCM}(s_1, s_2, s_5)$
   $s_1 = \langle T, S, C, Sp, L\rangle$
   $s_2 = \langle T, C, Sp\rangle$
   $s_5 = \langle T, C, S, Sp\rangle$
   Looking for maximal common subsequences:
   $\delta_{SCM}(s_1, s_2, s_5) = \{s \text{ match } \langle T, C, Sp\rangle\}$

4. $\delta_{SCM}(s_1, s_2, s_3, s_4, s_5)$
   The only subsequences that appear in all sequences:
   $\delta_{SCM}(s_1, s_2, s_3, s_4, s_5) = \{s \text{ match } \langle T\rangle, s \text{ match } \langle C\rangle\}$

5. $\delta_{SCM}(s_1, s_4)$
   $s_1 = \langle T, S, C, Sp, L\rangle$
   $s_4 = \langle C, T, Sp\rangle$
   Common subsequences that are maximal:
   $\delta_{SCM}(s_1, s_4) = \{s \text{ match } \langle T, Sp\rangle, s \text{ match } \langle C, Sp\rangle\}$

5.3 Exercise 3: NextPriorityConcept Algorithm Analysis

Calculate Candidate Subsets $A \subseteq M$

Starting from root concept predicates $\{s \text{ match } \langle T \rangle, s \text{ match } \langle C \rangle\}$, we need to generate candidate subgroups by adding elements:

For s match $\langle T, a \rangle$ or $\langle a, T \rangle$:

- s match $\langle T, C \rangle$: $\{s_1, s_2, s_3, s_5\}$
- s match $\langle T, S \rangle$: $\{s_1, s_5\}$
- s match $\langle T, Sp \rangle$: $\{s_1, s_2, s_5\}$
- s match $\langle T, L \rangle$: $\{s_1, s_3\}$

For s match $\langle C, a \rangle$ or $\langle a, C \rangle$:

- s match $\langle C, T \rangle$: $\{s_4\}$
- s match $\langle C, S \rangle$: $\{s_1, s_5\}$
- s match $\langle C, Sp \rangle$: $\{s_1, s_2, s_4, s_5\}$
- s match $\langle C, L \rangle$: $\{s_1, s_3\}$

Select Maximal Subgroups

From the candidate subsets, we select the maximal ones (those not contained in any other):

- $\{s_1, s_2, s_3, s_5\}$ from s match $\langle T, C \rangle$
- $\{s_1, s_2, s_5\}$ from s match $\langle T, Sp \rangle$
- $\{s_1, s_3\}$ from s match $\langle T, L \rangle$ and $\langle C, L \rangle$

Deduce the Concepts

For each maximal subgroup A, we calculate $(A, \delta_{SCM}(A))$:

- For $A = \{s_1, s_2, s_3, s_5\}$:
  - $(A, \{s \text{ match } \langle T, C \rangle\})$
- For $A = \{s_1, s_2, s_5\}$:
  - $(A, \{s \text{ match } \langle T, C, Sp \rangle\})$
- For $A = \{s_1, s_3\}$:
  - $(A, \{s \text{ match } \langle T, C, L \rangle\})$

These concepts form the immediate predecessors (children) of the root concept in the lattice hierarchy.

5.4 Exercise 4

Components of Complexity

- **Number of Subsets:** For a set M of n sequences, the total number of possible subsets is 2^n. This comes from the power set computation.

- **Computation for Each Subset:** For each subset $A \subseteq M$, we need to compute $\delta_{SCM}(A)$. Computing common subsequences is NP-complete. If we denote the complexity of solving the NP-complete problem as $O(2^p)$, where p is the problem size parameter, this adds significant complexity.

Total Complexity

The total complexity arises because we need to perform an NP-complete operation for each subset. Thus, the total complexity is $O(2^n \times 2^p)$, making it double exponential.

Practical Implications

This naive approach is computationally intractable even for small datasets. For example: - With just 5 sequences: $2^5 = 32$ subsets to examine - Each requiring an NP-complete computation.

This explains why more efficient algorithms like *NextPriorityConcept* are necessary.

5.5 Exercise 5

Root Concept Calculation

First, the root concept is calculated: - $(M, \delta_{SCM}(M) = \{s \text{ match } \langle T \rangle, s \text{ match } \langle C \rangle\})$ - M represents the full set of sequences $\{s_1, s_2, s_3, s_4, s_5\}$.

Candidate Subgroups Generation

For each predicate in $\delta_{SCM}(M)$, we add new elements $a \in M$:

1. From s match $\langle T \rangle$: - s match $\langle T, a \rangle$: Exploring extensions with each action:

- s match $\langle T, C \rangle$: $\{s_1, s_2, s_3, s_5\}$
- s match $\langle T, S \rangle$: $\{s_1, s_5\}$
- s match $\langle T, Sp \rangle$: $\{s_1, s_2, s_5\}$
- s match $\langle T, L \rangle$: $\{s_1, s_3\}$

2. From s match $\langle C \rangle$: - s match $\langle C, a \rangle$: Exploring extensions with each action:

- s match $\langle C, T \rangle$: $\{s_4\}$
- s match $\langle C, S \rangle$: $\{s_1, s_5\}$
- s match $\langle C, Sp \rangle$: $\{s_1, s_2, s_4, s_5\}$
- s match $\langle C, L \rangle$: $\{s_1, s_3\}$

Maximal Subgroups Selection
From these candidate subgroups, we select the maximal ones:

- $\{s_1, s_2, s_3, s_5\}$ from s match $\langle T, C\rangle$
- $\{s_1, s_2, s_5\}$ from s match $\langle T, C, Sp\rangle$
- $\{s_1, s_3\}$ from s match $\langle T, C, L\rangle$

Resulting Concepts
For these maximal subgroups, we calculate their complete concepts:

- $(\{s_1, s_2, s_3, s_5\}, \{s \text{ match } \langle T, C\rangle\})$
- $(\{s_1, s_2, s_5\}, \{s \text{ match } \langle T, C, Sp\rangle\})$
- $(\{s_1, s_3\}, \{s \text{ match } \langle T, C, L\rangle\})$

Verification with Lattice Diagram
Looking at Figure 1, we can verify that these concepts indeed appear as the immediate predecessors of the root concept (s_0, s_5), confirming our calculations match the hierarchical structure shown in the diagram.

5.6 Exercise 6

1. Calculating Predecessor Concepts Using Both Strategies
Starting Concept: $(s_1, s_3, \{s \text{ match } \langle T, C, L\rangle\})$
A. Using Naive Strategy (σ_{SN})
For σ_{SN}, we generate predicates by inserting a new element at any position in existing sequences:
Step 1: Generate Predicates From $\langle T, C, L\rangle$, possible insertions:

- Position 1: $\langle a, T, C, L\rangle$
- Position 2: $\langle T, a, C, L\rangle$
- Position 3: $\langle T, C, a, L\rangle$
- Position 4: $\langle T, C, L, a\rangle$

where $a \in \Sigma = \{T, S, C, Sp, L\}$.
Step 2: Find Maximal Subgroups Checking which sequences satisfy these predicates:

- For s_1: $\langle T, S, C, Sp, L\rangle$
- For s_3: $\langle T, C, L\rangle$

The maximal subgroup that satisfies any of these predicates would be: $(s_1, \{s \text{ match } \langle T, S, C, Sp, L\rangle\})$.
B. Using Augmented Strategy (σ_{SA})
For σ_{SA}, we only generate predicates of form $\langle x, a\rangle$ where $x \in \delta(A)$ and $a \in \Sigma$.
Step 1: Generate Predicates From $\delta(\{s_1, s_3\})$, we have $x = T, C, L$ and $a \in \{T, S, C, Sp, L\}$:

- $\langle T, S\rangle$: $\{s_1\}$
- $\langle T, C\rangle$: $\{s_1, s_3\}$
- $\langle T, Sp\rangle$: $\{s_1\}$
- $\langle T, L\rangle$: $\{s_1, s_3\}$
- $\langle C, S\rangle$: $\{s_1\}$
- $\langle C, L\rangle$: $\{s_1, s_3\}$

Step 2: Find Maximal Subgroups The maximal subgroup is: $(s_1, \{s \text{ match } \langle T, S, C, Sp, L\rangle\})$.
2. Concept Lattice Using Augmented Strategy (σ_{SA})
The augmented strategy generates a smaller lattice:
Level 1 (Root): $(s_1, s_2, s_3, s_4, s_5, \{s \text{ match } \langle T\rangle, s \text{ match } \langle C\rangle\})$
Level 2:

- $(s_1, s_2, s_5, \{s \text{ match } \langle T, C, Sp\rangle\})$
- $(s_1, s_3, \{s \text{ match } \langle T, C, L\rangle\})$

Level 3 (Leaves):

- $(s_1, \{s \text{ match } \langle T, S, C, Sp, L\rangle\})$
- $(s_2, \{s \text{ match } \langle T, C, Sp\rangle\})$
- $(s_3, \{s \text{ match } \langle T, C, L\rangle\})$
- $(s_4, \{s \text{ match } \langle C, T, Sp\rangle\})$
- $(s_5, \{s \text{ match } \langle T, C, S, Sp\rangle\})$

The augmented strategy σ_{SA} produces a more compact lattice while maintaining the essential concept relationships.

5.7 Exercise 7

Initial Data Set

$$\begin{aligned}
s_1 &= \langle T, S, C, Sp, L\rangle \\
s_2 &= \langle T, C, Sp\rangle \\
s_3 &= \langle T, C, L\rangle \\
s_4 &= \langle C, T, Sp\rangle \\
s_5 &= \langle T, C, S, Sp\rangle
\end{aligned}$$

Lattice Construction Using δ_{SCP} and σ_{SA}
1. Root Concept First, calculate prefix common subsequences for all sequences:

- $A = \{s_1, s_2, s_3, s_4, s_5\}$
- $\delta_{SCP}(A) = \{s \text{ match } \langle T \rangle\}$, since $\langle T \rangle$ is the only common prefix except for s_4.

2. Level 2 Concepts Using augmented strategy σ_{SA} to generate predicates from the root:

For sequences starting with T (s_1, s_2, s_3, s_5):

- $\delta_{SCP}(\{s_1, s_2, s_3, s_5\}) = \{s \text{ match } \langle T, C \rangle\}$

For the sequence starting with C (s_4):

- $\delta_{SCP}(\{s_4\}) = \{s \text{ match } \langle C, T \rangle\}$

3. Level 3 Concepts From $\{s \text{ match } \langle T, C \rangle\}$, generate the next level:

1. Group $\{s_1, s_2, s_5\}$:
   - $\delta_{SCP}(\{s_1, s_2, s_5\}) = \{s \text{ match } \langle T, C, Sp \rangle\}$
2. Group $\{s_1, s_3\}$:
   - $\delta_{SCP}(\{s_1, s_3\}) = \{s \text{ match } \langle T, C, L \rangle\}$

4. Leaf Concepts (Individual Sequences)

$$\begin{aligned}
\delta_{SCP}(\{s_1\}) &= \{s \text{ match } \langle T, S, C, Sp, L \rangle\} \\
\delta_{SCP}(\{s_2\}) &= \{s \text{ match } \langle T, C, Sp \rangle\} \\
\delta_{SCP}(\{s_3\}) &= \{s \text{ match } \langle T, C, L \rangle\} \\
\delta_{SCP}(\{s_4\}) &= \{s \text{ match } \langle C, T, Sp \rangle\} \\
\delta_{SCP}(\{s_5\}) &= \{s \text{ match } \langle T, C, S, Sp \rangle\}
\end{aligned}$$

Complete Lattice Structure

1. **Root Level:** $(s_1, s_2, s_3, s_4, s_5, \{s \text{ match } \langle T \rangle\})$
2. **Second Level:**
   - $(s_1, s_2, s_3, s_5, \{s \text{ match } \langle T, C \rangle\})$
   - $(s_4, \{s \text{ match } \langle C, T \rangle\})$
3. **Third Level:**
   - $(s_1, s_2, s_5, \{s \text{ match } \langle T, C, Sp \rangle\})$
   - $(s_1, s_3, \{s \text{ match } \langle T, C, L \rangle\})$
4. **Leaf Level:**
   - $(s_1, \{s \text{ match } \langle T, S, C, Sp, L \rangle\})$

- $(s_2, \{s \text{ match } \langle T, C, Sp\rangle\})$
- $(s_3, \{s \text{ match } \langle T, C, L\rangle\})$
- $(s_4, \{s \text{ match } \langle C, T, Sp\rangle\})$
- $(s_5, \{s \text{ match } \langle T, C, S, Sp\rangle\})$

Key Observations

- The prefix description δ_{SCP} creates a simpler lattice structure compared to maximal common subsequences (SCM).
- The focus on prefixes reduces the number of possible combinations.
- The augmented strategy σ_{SA} helps maintain a more compact structure while preserving essential relationships.
- The resulting lattice is more efficient to compute while still capturing the important sequence patterns.

5.8 Exercise 8

Let's analyze the description predicates δ_{SDCM} for each subgroup and interpret their meaning.

For {s1, s3}

$$\delta_{SDCM}(\{s1, s3\}) = \{s \text{ match } T[\text{1-3}]C[\text{3-5}]L\}$$

Interpretation:

- Work (T) occurs within hours 1-3 (8:00-10:00).
- Coffee (C) follows 3-5 hours later (11:00-16:00).
- Reading (L) occurs after coffee.
- This indicates a pattern of morning work, followed by a coffee break, then reading.

For {s1, s2, s3, s5}

$$\delta_{SDCM}(\{s1, s2, s3, s5\}) = \{s \text{ match } T[\text{1-2}]C[\text{2-3}]\}$$

Interpretation:

- Work (T) starts within hours 1-2 (8:00-9:00).
- Coffee (C) follows 2-3 hours after starting work.
- This suggests a common morning routine: early work followed by mid-morning coffee.

For {s1, s4}

$$\delta_{SDCM}(\{s1, s4\}) = \{s \text{ match } C[\text{1-8}]Sp\}$$

Interpretation:

- Coffee (C) occurs within hours 1-8 (9:00-16:00).
- Sport (Sp) follows after coffee.
- This shows a pattern of having coffee sometime during the day followed by sports activity.

For {s1, s4, s5}

$$\delta_{SDCM}(\{s1, s4, s5\}) = \{s \text{ match } C[\text{2-8}]Sp\}$$

Interpretation:

- Coffee (C) occurs within hours 2-8 (10:00-16:00).
- Sport (Sp) follows after coffee.
- This is similar to the previous pattern but with a more restricted coffee time window.
- It shows a common pattern of mid-day coffee followed by sports.

Key Observations:

1. **The temporal dimension adds richness to the analysis:**
    - Can identify not just common activities but also common timing patterns.
    - Shows how activities are sequenced relative to each other.
2. **The distance-based approach allows for flexibility:**
    - Activities don't need to occur at exactly the same time.
    - Time windows capture natural variations in daily routines.
3. **Patterns reveal different types of daily routines:**
    - Morning work-coffee-reading routine.
    - Coffee followed by sports pattern.
    - Early work followed by coffee pattern.

5.9 Exercise 9

Most Common Length-2 Patterns:

Morning Patterns:

- Wakeup → Breakfast
- Breakfast → Work

Mid-Day Patterns:

- Work → Lunch
- Lunch → Coffee/Work
- Coffee → Work

Evening Patterns:

- Work/Sports → Dinner
- Dinner → Sleep/Rest: 52%

Other Notable Patterns:

- Work → Coffee
- Lunch → Nap
- Sports → Dinner

The Clear Daily Routines:

1. **Morning Sequence:** Wakeup → Breakfast → Work
2. **Mid-Day Sustenance:** Work → Lunch
3. **Evening Wind-Down:** Activity → Dinner → Sleep

5.10 Exercise 10

1. **Simple Maximal Subsequences (Length-2):**
   - **Predicates:** 15-20 basic patterns (e.g., `<Wakeup, Breakfast>`, `<Breakfast, Work>`)
   - **Concepts:** Around 30-40 unique combinations
   - **Execution Time:** Very fast (milliseconds)
2. **Complete Distance with 'NaiveDistance':**
   - **Predicates:** 50-60 temporal patterns
   - **Concepts:** Around 80-100 unique combinations

- **Execution Time:** Moderate (seconds)

3. **Complete Distance with ‘CompleteDistance‘:**
   - **Predicates:** Over 100 temporal patterns
   - **Concepts:** 150+ unique combinations
   - **Execution Time:** Longest (several seconds)

5.11 Exercise 11

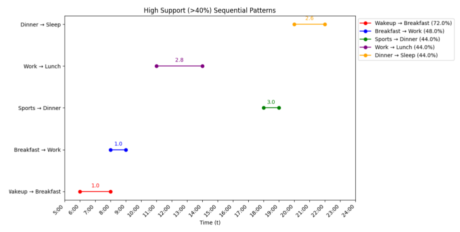


Figure 38: Exercise 11

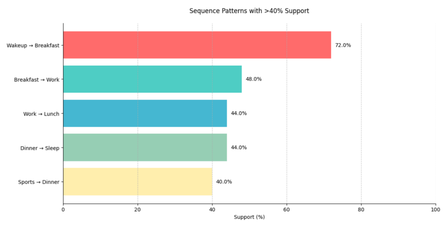


Figure 39: Exercise 11