

1. The chart appears to be showing data on CpG island hypermethylation in various human tissues or organs in the context of cancer. The types of data are likely to be categorical (the different organs or tissues) and quantitative (the percentage of hypermethylation).
2. The author uses:

- Bars (Marks): To represent individual data points, which in this case are the methylation percentages for each category.

- Height (Channel): The height of each bar encodes the quantitative methylation percentage.

- Position: The placement of the bars on the x and y axes represents the categorical information of cell type/tissue and indicators/genes.

- Color/Shading: It seems there are different colors or shades, possibly to differentiate between various categories or to represent another variable.

1. Visualization Problems:

- Readability: 3D bar charts can be visually complex and difficult to read, making it hard to compare bars that are farther away.

- Overplotting: With many bars in a 3D space, it's possible for some data to be obscured by others when viewed from certain angles.

- Lack of Clear Labels: The axis labels may be hard to read due to the angle and the density of information.

- Color use: If the bars are color-coded, the use of shading due to 3D rendering can mislead interpretations of the data.

1. Redesign:

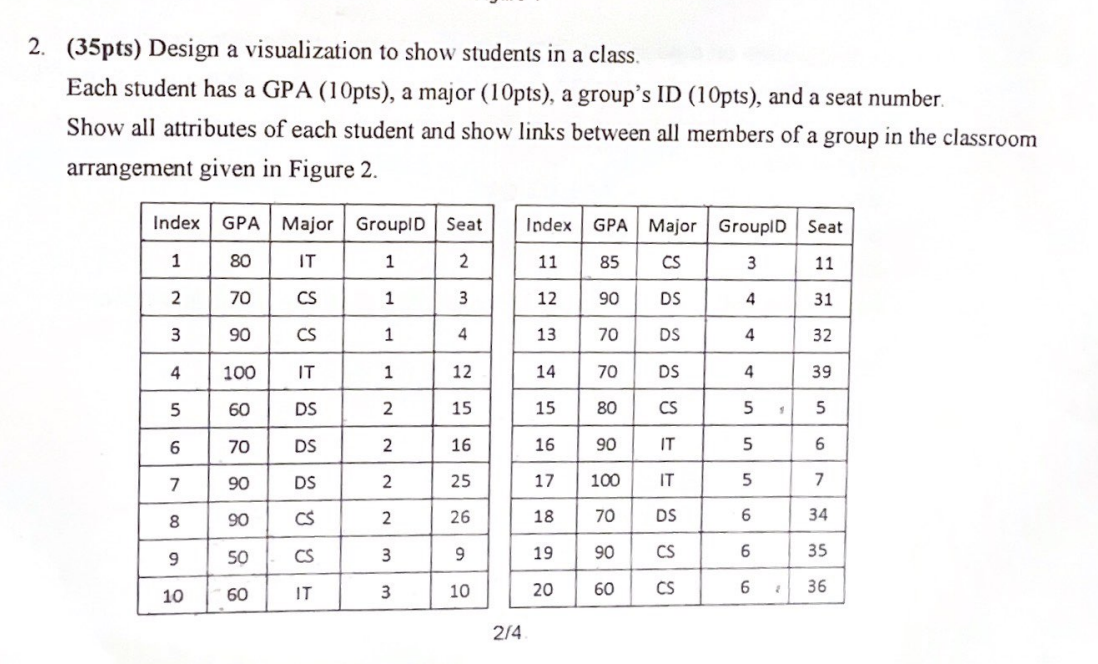
- Switch to a 2D Format: Use a grouped or stacked bar chart, or multiple panels (small multiples) for clarity.

- Simplify: Only show key comparisons to avoid overplotting.

- Enhance Readability: Ensure all labels are legible and axes are clearly marked.

- Consistent and Clear Color Scheme: Use color judiciously to differentiate data categories, ensuring there's a legend and it is colorblind-friendly.

- Interactive Elements: If possible, an interactive chart where you could hover for more details would help manage the complexity of the data.





import matplotlib.pyplot as plt

GPA = [80,70,90,100,60,70,90,90,50,60,85,90,70,70,80,90,100,70,90,60]

GroupID = [1,1,1,1,2,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6]

major = ['IT', 'CS', 'CS', 'IT', 'DS', 'DS', 'DS', 'CS', 'CS', 'IT', 'CS', 'DS', 'DS', 'DS', 'CS', 'IT', 'IT', 'DS', 'CS', 'CS']

Seat = [2,3,4,12,15,16,25,26,9,10,11,31,32,39,5,6,7,34,35,36]

fig, ax = plt.subplots()

for i in range(len(GPA)):

ax.scatter(Seat[i], GPA[i], c='b')

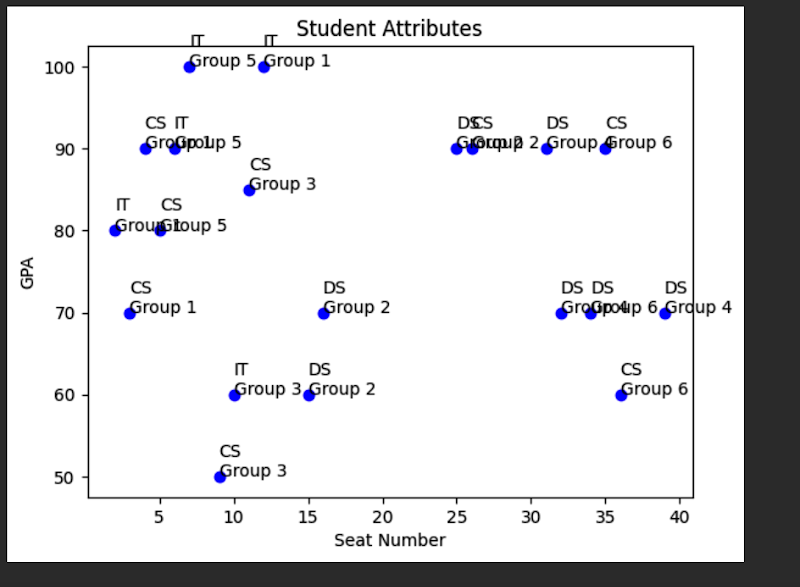
ax.text(Seat[i], GPA[i], f'{major[i]}\nGroup {GroupID[i]}')

ax.set\_xlabel('Seat Number')

ax.set\_ylabel('GPA')

ax.set\_title('Student Attributes')

plt.show()



Student in a class

- GPA: [Magnitude Channels: Ordered Attributes] Color Saturation

- GroupID (6 class): [Identity Channels: Categorical Attributes] Color Hue

Hai attribute này đều có thể dùng màu để thể hiện. Tụi m chọn một trong hai

- Major (3 class): [Identity Channels: Categorical Attributes] Shape

- Seat: Position (as in Figure 2)

* Link for girouD: Manipulate view. CHọn một seat -> hiện những bạn cùng nhóm -> ẩn những bạn khác nhóm

Allowing users to select one or more elements of interest in a vis is a fundamental action that supports nearly every interactive idiom. The output of a selection operation is often the input to a subsequent operation. In particular, the change choice is usually dependent on a previous select result. (Đọc trong sách thêm)

import numpy as np

import matplotlib.pyplot as plt

GPA = [80,70,90,100,60,70,90,90,50,60,85,90,70,70,80,90,100,70,90,60]

GroupID = [1,1,1,1,2,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6]

major = ['IT', 'CS', 'CS', 'IT', 'DS', 'DS', 'DS', 'CS', 'CS', 'IT', 'CS', 'DS', 'DS', 'DS', 'CS', 'IT', 'IT', 'DS', 'CS', 'CS']

Seat = [2,3,4,12,15,16,25,26,9,10,11,31,32,39,5,6,7,34,35,36]

fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))

for group\_id in set(GroupID):

group\_members = [seat for seat, group in zip(Seat, GroupID) if group == group\_id]

row = (group\_id - 1) // 2

col = (group\_id - 1) % 2

ax = axs[row][col]

ax.set\_xticks(np.arange(1, 7))

ax.set\_yticks(np.arange(1, 8))

ax.set\_xticklabels(np.arange(1, 7))

ax.set\_yticklabels(np.arange(1, 8))

ax.set\_title(f'Group {group\_id}')

for member\_seat in group\_members:

row = (member\_seat - 1) // 6

col = (member\_seat - 1) % 6

ax.add\_patch(plt.Rectangle((col, row), 1, 1, fill=False, edgecolor='b', lw=2))

plt.tight\_layout()

plt.show()

ANOTHER

// This is a simplified version of your data structure in JSON

const students = [

// ... your data

];

// Classroom layout for 40 seats in a 5x8 grid

const seatPositions = [

// {seat: 1, x: 10, y: 10}, {seat: 2, x: 50, y: 10}, ... (fill out the rest according to Figure 2 layout)

];

// Create SVG

const svg = d3.select('body').append('svg')

.attr('width', 800)

.attr('height', 600);

// Draw seats

svg.selectAll('rect')

.data(seatPositions)

.enter().append('rect')

.attr('x', d => d.x)

.attr('y', d => d.y)

.attr('width', 40)

.attr('height', 40)

.attr('stroke', 'black')

.attr('fill', 'white');

// Place students in seats

svg.selectAll('text')

.data(students)

.enter().append('text')

.attr('x', d => seatPositions[d.seat - 1].x) // You'll need to adjust these indices to match your layout

.attr('y', d => seatPositions[d.seat - 1].y)

.text(d => `GPA: ${d.GPA}, Major: ${d.Major}`)

.attr('font-size', '10px');

// Draw group links

// This function finds the x, y position of a seat by number

const findPosition = (seatNumber) => {

return seatPositions.find(pos => pos.seat === seatNumber);

};

// For each group, find the members and draw lines between them

students.forEach(student => {

students.forEach(peer => {

if (student.GroupID === peer.GroupID && student.Index < peer.Index) { // To avoid duplicate lines

const startPos = findPosition(student.Seat);

const endPos = findPosition(peer.Seat);

svg.append('line')

.attr('x1', startPos.x + 20) // +20 to draw line from center of seat

.attr('y1', startPos.y + 20)

.attr('x2', endPos.x + 20)

.attr('y2', endPos.y + 20)

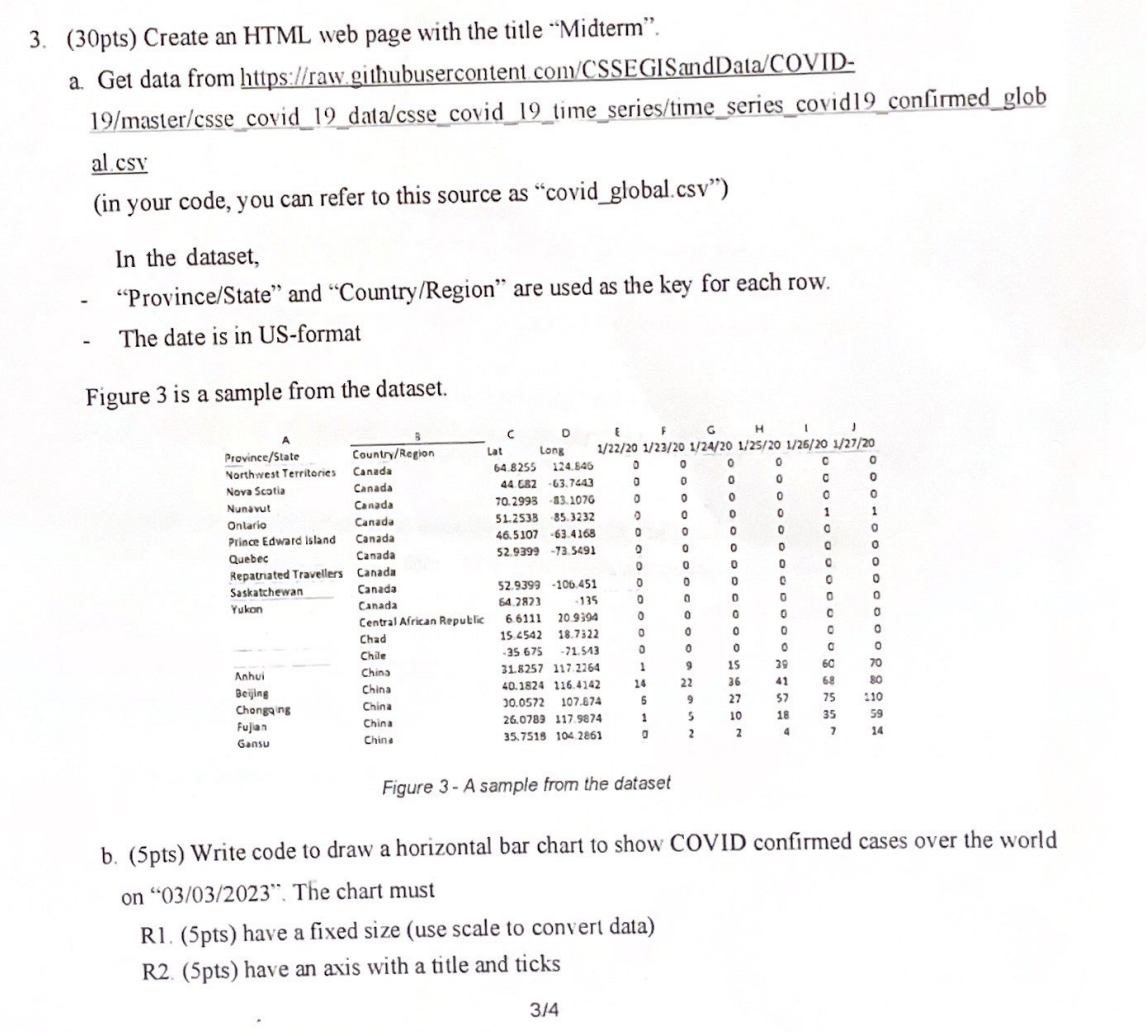
.attr('stroke', 'red')

.attr('stroke-width', 2);

}

});

});





// Define the URL for the COVID-19 data

const dataUrl = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse\_covid\_19\_data/csse\_covid\_19\_time\_series/time\_series\_covid19\_confirmed\_global.csv';

// Use D3 to fetch the data from the provided URL

d3.csv(dataUrl).then(function(data) {

console.log(data);

}).catch(function(error) {

console.error(error);

});

b. Answer

