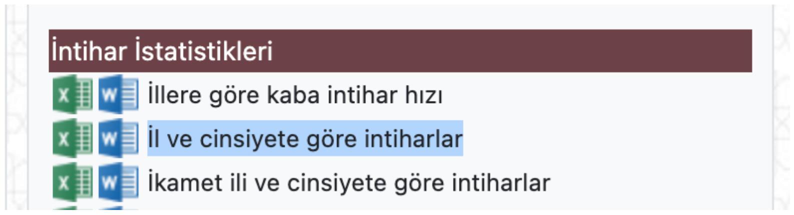
CS405 Computer Graphics
Assignment 1

Barış Ulaş Çukur
29461

The dataset

I've chosen to work with suicide rates by cities and genders table. The file is called "il ve cinsiyete gore intiharlar.xls" and it is available on the website with the label "İl ve cinsiyete göre intiharlar".



İntihar İstatistikleri

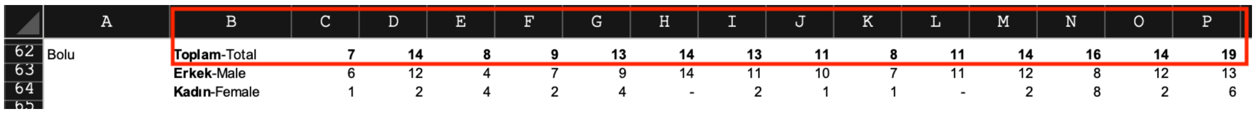
İllere göre kaba intihar hızı

İl ve cinsiyete göre intiharlar

İkamet ili ve cinsiyete göre intiharlar

The excel sheet that the code expects to be uploaded.

Here's a quick summary of the dataset and the subset I've chosen to work with.



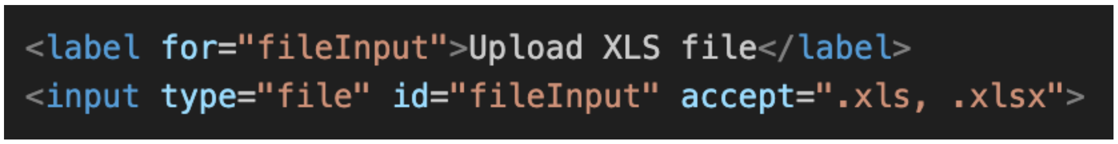
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
62	Bolu	**Toplam**-Total	**7**	**14**	**8**	**9**	**13**	**14**	**13**	**11**	**8**	**11**	**14**	**16**	**14**	**19**
63		**Erkek**-Male	6	12	4	7	9	14	11	10	7	11	12	8	12	13
64		**Kadın**-Female	1	2	4	2	4	-	2	1	1	-	2	8	2	6

The subset of the suicide rates by cities and genders table that I've chosen to work with.

The subset I've chosen to work with was about the overall suicide rates by all genders in Bolu between 2009 and 2022. The dataset included year information on the 5th row of the table and the overall suicide rates summary in Bolu on the 62nd row.

Reading the data

The code includes a *fileInput* label for selecting an *xls* file from the local storage, more specifically *"il ve cinsiyete gore intiharlar.xls"* file as the code was hardcoded to work with the rows and columns of this specific file. After the file has been selected, the website assigns it an *id* called "fileInput". Later on, the other parts of the code are able to reference this file using this *id*.



```
<label for="fileInput">Upload XLS file</label>
<input type="file" id="fileInput" accept=".xls, .xlsx">
```

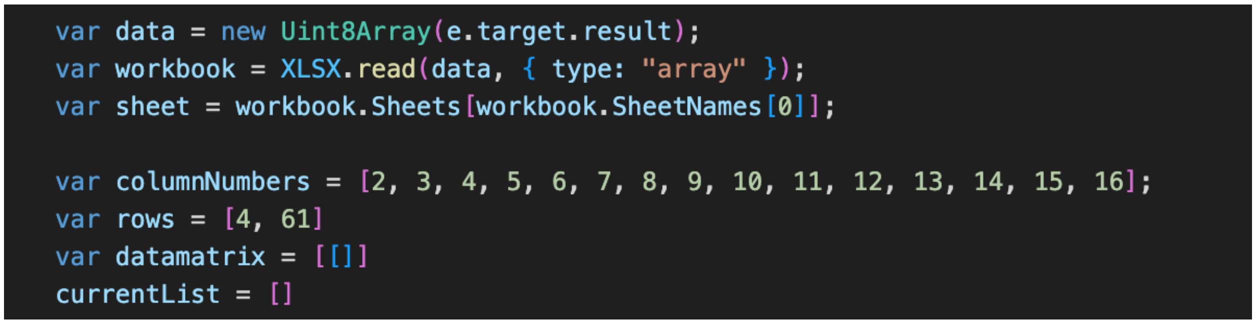
Label prompts the user to upload an xls file from the local storage and input field assigns it "fileInput" id.

Filtering the data

While *JavaScript* works fine with *CSV* files, the dataset was in *xls (excel datasheet)* format. To work with the file by parsing it, I needed a library to handle the reading process. After researching about it, I've decided to use *SheetJS* library. The code is able to access the specific rows and columns of the excel sheet using this library.

The code works by reading the *xls* file and storing the resulting object in the *sheet* variable. Later, code iterates over each row and column of the file. When it reaches the rows and columns of the specified parts in the *columnNumbers* and *rows* variables, it appends the data inside the *currentList* variable. After the loop reaches the end of the current row, it appends the *currentList* inside the *datamatrix* variable.

In the end, we end up with a matrix that consists of the data filtered by the *columnNumbers* and *rows* lists.



```
var data = new Uint8Array(e.target.result);
var workbook = XLSX.read(data, { type: "array" });
var sheet = workbook.Sheets[workbook.SheetNames[0]];

var columnNumbers = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16];
var rows = [4, 61]
var datamatrix = [[]]
currentList = []
```

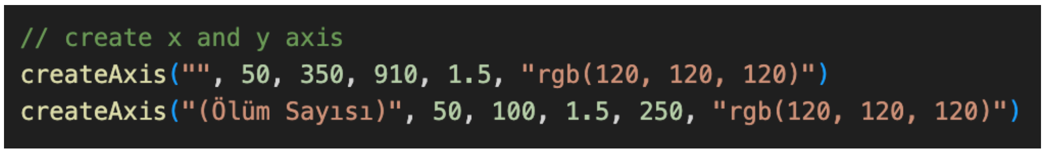
JS code accessing the certain columns and rows of the uploaded and parsed excel sheet to get the data.

Visualizing the data

- **Creating the axes**

  Before visualizing the data, I needed to add an *SVG* container with dimensions 1000 to 600 pixels to place the graph in. Data visualization is handled by the two functions of the code called *createAxis* and *createLabel*.

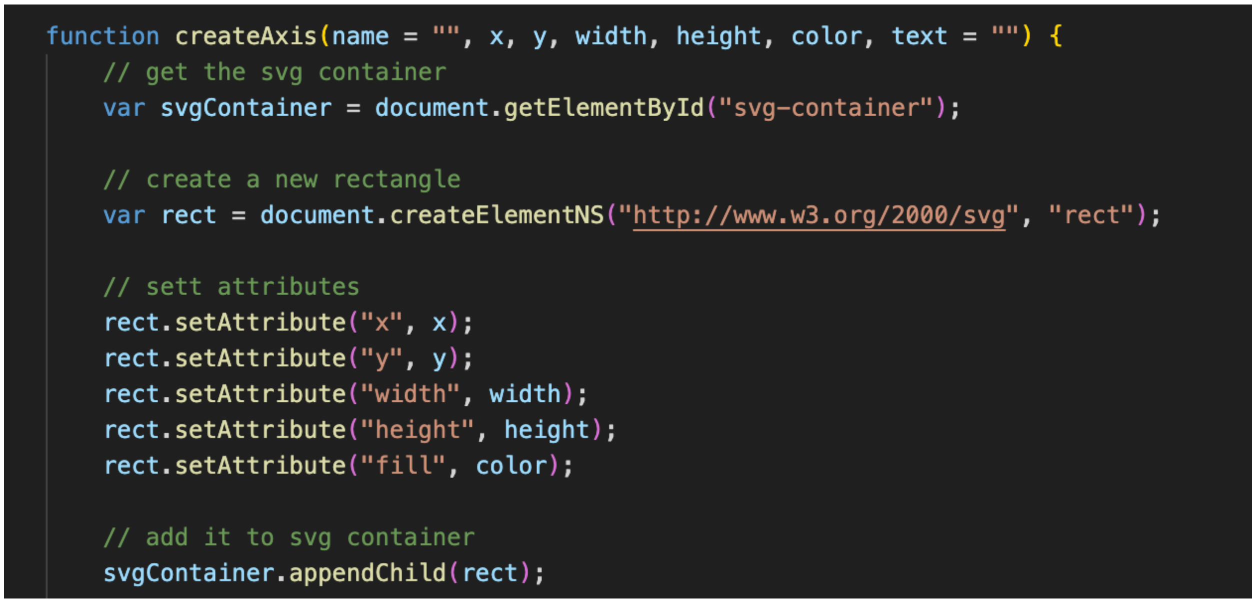
  createAxis is responsible for creating the *x* and *y axes* of the graph. It takes *name, x, y, width, height, color,* and *text* as parameters and creates an *axis* accordingly.



```
// create x and y axis
createAxis("", 50, 350, 910, 1.5, "rgb(120, 120, 120)")
createAxis("(Ölüm Sayısı)", 50, 100, 1.5, 250, "rgb(120, 120, 120)")
```

createAxis function being called to create x and y axes.

The *createAxis* function works by creating a rectangle of specified *width* and *height* integers from the function parameter. Similarly, it also places the rectangle to *x* and *y* coordinates given by the parameter of the function.



```
function createAxis(name = "", x, y, width, height, color, text = "") {
    // get the svg container
    var svgContainer = document.getElementById("svg-container");

    // create a new rectangle
    var rect = document.createElementNS("http://www.w3.org/2000/svg", "rect");

    // sett attributes
    rect.setAttribute("x", x);
    rect.setAttribute("y", y);
    rect.setAttribute("width", width);
    rect.setAttribute("height", height);
    rect.setAttribute("fill", color);

    // add it to svg container
    svgContainer.appendChild(rect);
```

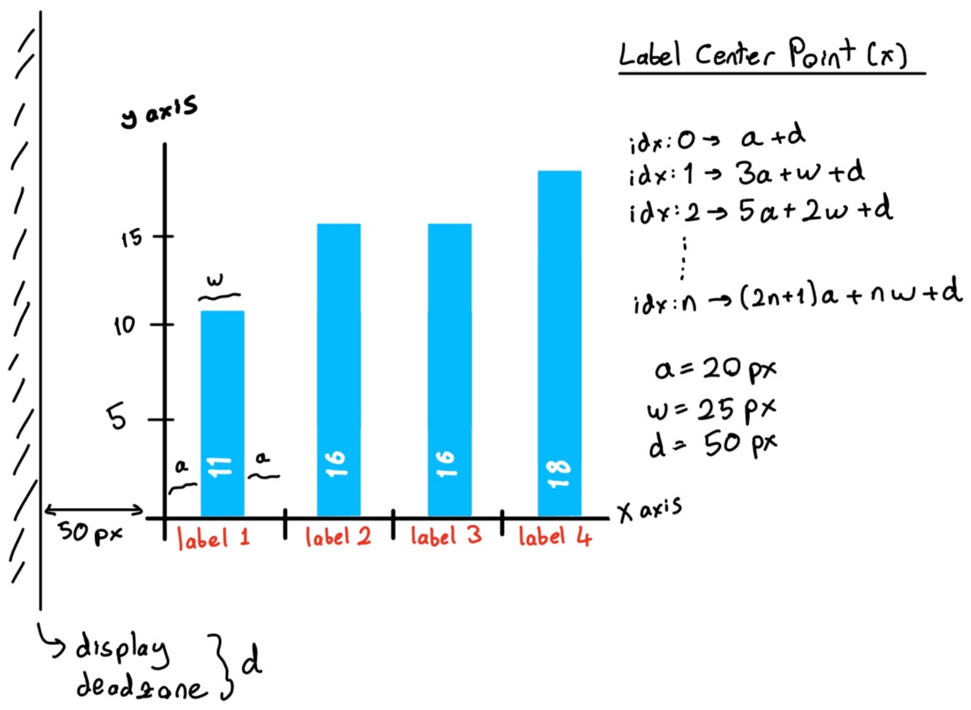
createAxis function creating a rectangle to be used as an x or y axis.

- **Adding labels to axes**

createLabel function is responsible for the creation of the axis labels, such as *0, 10, 20* for *y-axis* and *2009, 2010, 2011* for the *x-axis*. The function takes *type, x, y,* and *text* variables as parameters.

It basically works by adding a *text* element to the specified position by the function caller. It also adds a small vertical rectangle in the shape of a line “|” 20 pixels to the left of the text.

The logic of adding labels dynamically to the graph required to devise a generalized formula for the *x-axis* offset of the labels. Instead of explaining it in a long text, I’ll go over the visualization I’ve created.



Alignment of x-axis labels with respect to their indices.

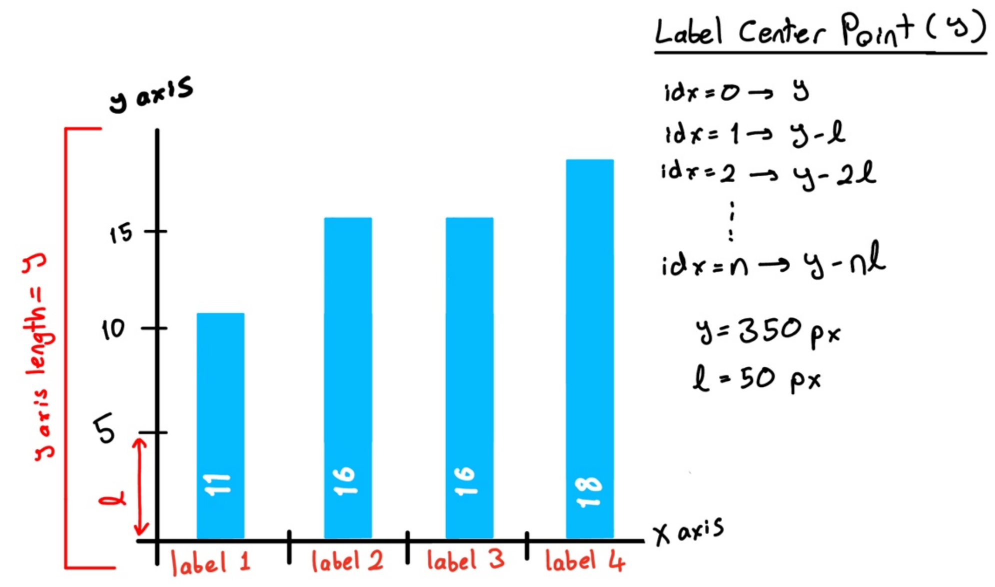
Basically, the generalized formula for the *x-axis offset* of the *label n* at *index n* is calculated using:

x_axis_offset = (2*n* + 1) * *a* + *n* * *w* + *d*

n: *Index of the label*
a: *offset of the bar graph from the left = 20px*
w: *width of the bar graph = 25px*
d: *display deadzone = 50px*

For example, the second label of the *x-axis* with the value “label 2” at index 1 would have an *x-axis* offset of: (2*2 + 1)*20 + 2*25 + 50 = 200px. This generalized formula ensures that each bar graph labels are placed correctly in their places with respect to their indices.

Adding labels to *y-axis* required to devise a formula similar to the one that was devised for the labels on *x-axis*. A basic visualization was created again to explain the thought and the development process.



Alignment of y-axis labels with respect to their indices.

Similarly, I've devised a generalized formula for the alignment of *y-axis* labels with respect to their indices.

*y_axis_offset = y – n*l*

y: length of the y-axis = 350px
n: index
l: the space between two labels = 50px

For example, the second element of the graph with the value 10 and index 1 would have a *y-axis* offset of: 350 – 1*50 = 300px.

- **Creating the bar charts**

For the creation of the bar charts, *createAxis* function is repurposed and used again, but called a bit differently this time.

```
for (var i = 0; i < datamatrix[2].length; i++) {
    x_offset = ((2 * i + 1) * 20) + ((i) * 25) + 50
    createAxis("", x_offset, 350 - datamatrix[2][i] / 5 * 50, 25, datamatrix[2][i] / 5 * 50, "rgb(20, 143, 204)", datamatrix[2][i])
}
```

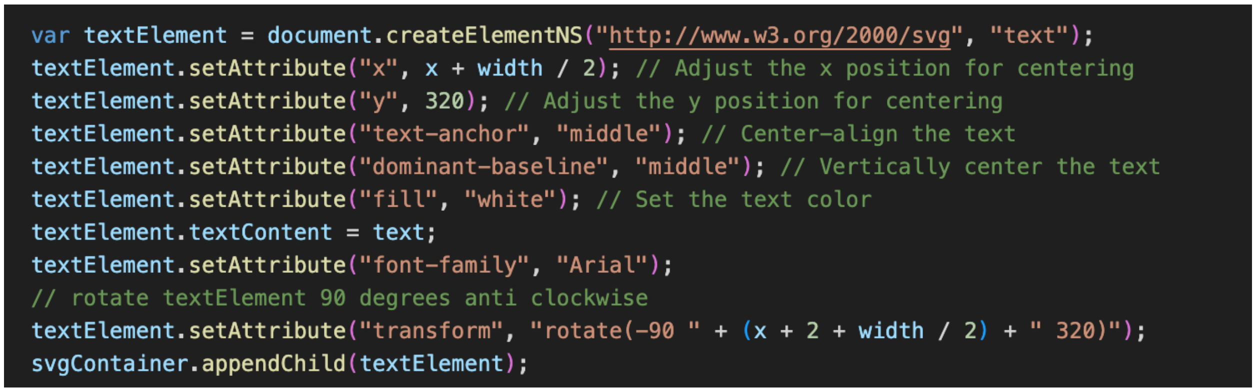
Similar to the previous offset calculations, I calculated the required offsets for the *x-axis* locations of the bar elements. The offset formula for calculating the offset here is the same that was used for calculation of the *x-axis* labels, so I won't go into details of the calculation again.

What’s different here is the calculation of the bar graph (a rectangle) height with respect to *y-axis* offset. As JavaScript starts drawing the rectangle from the top, I needed to move the draw point to the place where the remaining part of the rectangle would be equal to the value it holds.

A basic formula for this approach could be devised as follows:

Starting point of the rectangle (*y*) = *y-axis* length – value of the rectangle

This function is also responsible for adding the value of the current element inside the rectangle but rotated 90 degrees anti-clockwise.



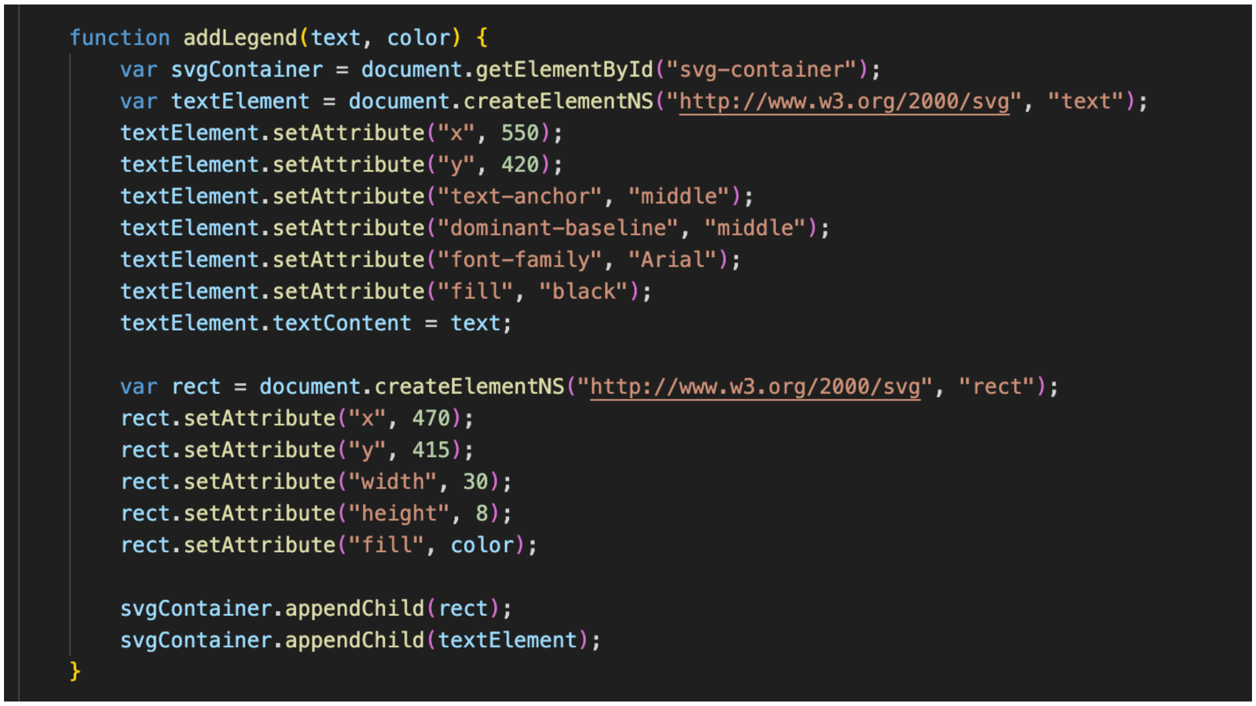
```javascript
var textElement = document.createElementNS("http://www.w3.org/2000/svg", "text");
textElement.setAttribute("x", x + width / 2); // Adjust the x position for centering
textElement.setAttribute("y", 320); // Adjust the y position for centering
textElement.setAttribute("text-anchor", "middle"); // Center-align the text
textElement.setAttribute("dominant-baseline", "middle"); // Vertically center the text
textElement.setAttribute("fill", "white"); // Set the text color
textElement.textContent = text;
textElement.setAttribute("font-family", "Arial");
// rotate textElement 90 degrees anti clockwise
textElement.setAttribute("transform", "rotate(-90 " + (x + 2 + width / 2) + " 320)");
svgContainer.appendChild(textElement);
```

createAxis function adding the value of the current element inside the rectangle.

If needed, *createAxis* function also takes a non-required *name* parameter. This is to be used as the label of the *y-axis*, “Ölüm Sayısı” for this instance. If a name is provided, the function adds it to the top of the *y-axis* by creating a new text element with the *x-axis* value of the SVG container and *y-axis* value of the *y-axis* length - 20.

- **Adding a legend**

Lastly, I added a legend to the graph to visualize the kind of the value shown on the graph. To create the label, I just added a simple text and rectangle elements side by side at the center of the graph and a little bit below it.

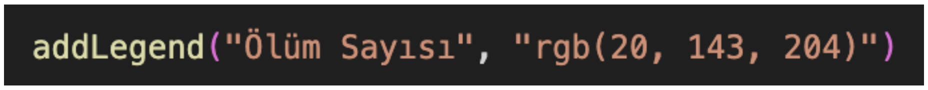


```
function addLegend(text, color) {
    var svgContainer = document.getElementById("svg-container");
    var textElement = document.createElementNS("http://www.w3.org/2000/svg", "text");
    textElement.setAttribute("x", 550);
    textElement.setAttribute("y", 420);
    textElement.setAttribute("text-anchor", "middle");
    textElement.setAttribute("dominant-baseline", "middle");
    textElement.setAttribute("font-family", "Arial");
    textElement.setAttribute("fill", "black");
    textElement.textContent = text;

    var rect = document.createElementNS("http://www.w3.org/2000/svg", "rect");
    rect.setAttribute("x", 470);
    rect.setAttribute("y", 415);
    rect.setAttribute("width", 30);
    rect.setAttribute("height", 8);
    rect.setAttribute("fill", color);

    svgContainer.appendChild(rect);
    svgContainer.appendChild(textElement);
}
```

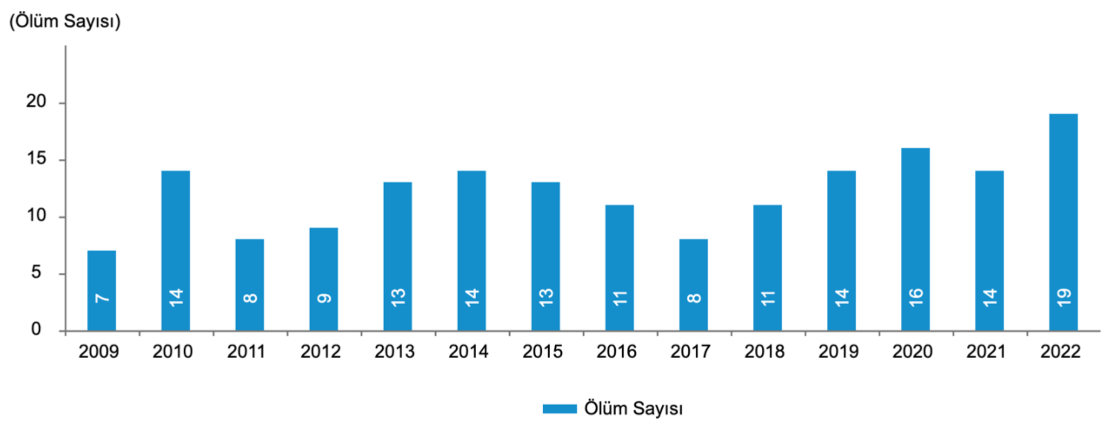
addLegend function adding a legend to the graph.



```
addLegend("Ölüm Sayısı", "rgb(20, 143, 204)")
```

The function call of the addLegend function to create a legend with specified text color.

Finally, we end up with a graph that has correctly aligned *x* and *y* labels combined with accurate bar lengths including their values inside, rotated 90 degrees anti-clockwise.



The final graph we end up with.

Final notes

After completing the homework, I realized that parsing the Excel sheet was unnecessary, and I could have manually copied the cell values into a list or a matrix. While the code I submitted may not be 100% dynamic in terms of creating a graph for different excel sheets, I don't believe that was the primary focus of this assignment anyway. As previously stated, the code is able to generate an accurate graph if *"il ve cinsiyete gore intiharlar.xls"* file is inputted using the file picker dialog. Moreover, the values of the specified cells can be modified to generate a different graph if needed.