PRA3006 - Group 1

By Sanne Aarts,
Aleksandrs Handramajs,
Véronique Hehl

Presentation outline

1

Introduction

Research question and its relevance

2

Methods

Webservices

Queries

Execution of the code

3

Discussion

Limitations and Advantages

Problems and solutions

4

Conclusion

Goals achieved

Future Outlook

Introduction

Research Question

Simultaneous usage affects effectiveness or may cause other adverse effects [2].
By focussing only on significant drug interactions the data in the wikidata database suffices and thus provides a **natural scope** to this project.

The primary component of a product that is responsible for producing the intended result.
By focussing on active ingredients only, we **group together** as many brands as possible while still focussing on the part that are **most likely to interact.**

How can we visualize the **significant drug interactions** between **active ingredients** of medications used to treat **mental illnesses** in an **accessible** manner?

Broad range of mental health disorders,
The combination of **COVID-19** and the **overburdened mental healthcare system** has created a lot of mental health problems, especially among gen-Z **[1]** and at the same time made **trustworthy information inaccessible**. By focussing only on mental health disorders we wish to create a resource that is **informative** yet not so **overwhelming** that it takes away from its efficiency

To be used by people of all abilities.
Mental disorders may tend to go together and patients suffering from a combination may be prompted to combine medicine. Whilst generally approvement or disapprovement by their therapist is required, it is important that **data on interactions** can be found in a **readable manner.** We realize that anyone may suffer from mental illness and by focussing on **accessibility** wish to reach as many people as possible.

[1,2]

Methods

Web Services

The Data

A central data storage which can be accessed through the SPARQL endpoint.

The way way in which wikidata structures its data offers an increased amount of freedom in comparison to for example relational databases

The Diagram

D3.js is a JavaScript library that aids the visualization of data.

Allows us to apply yet customize their prebuilt visualizations to our own data. Inclusion of math helper functions helped us improve scalability

The Design

CSS framework based on utility classes.

Simplifies the maintenance and scaling of code whilst allowing us the focus on functionality over design.

The Dynamics

JavaScript library which facilitates manipulation of HTML.

Simplifies event handling and allowed us to create responsive elements which adapt to the size of the data they contain.

The Queries

Disease query

Getting the diseases

Old

```
SELECT DISTINCT ?disease ?diseaseLabel ?typeLabel
WHERE {
  {
    VALUES ?item {wd:Q12135} #all entries of "mental disorder
          ?type wdt:P279 ?item . # "categories" of mental disorders, subclasses of mental disorders
          ?disease wdt:P279 ?type . #the mental disorders which fall into that category
  }
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
}
```

New

```
SELECT DISTINCT ?disease ?diseaseLabel
  WHERE {
    {
      VALUES ?item {wd:Q112193867} #all classes of disease (its instances are classes such as ALS)
            ?disease wdt:P31 ?item ; #disease is an instance of class of disease
                     wdt:P1995 wd:Q7867 ; #where the health specialty is psychiatry
                     wdt:P2176 ?treatment . #and the disease has any form of treatment
      }
      SERVICE wikibase:label {bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
  }
```

Main differences:

- We now check for diseases which fall under psychiatry, which has a more clear hierarchy of information.
- We now filter out diseases which do not have any treatment.

Medication query

Getting the Medications

Old

```
SELECT DISTINCT ?medicine ?medicineLabel ?type ?typeLabel ?interactswithLabel ?treatsLabel
WHERE {
  {
      VALUES ?item {wd:Q12140} #selects all entries that are medicines
        ?medicine wdt:P279 ?item ; #?medicine is a subclass of those entries
             wdt:P2175 wd:Q181923 ; #which is selected only if the medical condition treated is Attention deficit Hyperactivity disorder
             wdt:P769 ?interactswith . #Significant drug interactions of that medicine
        ?type wdt:P31 ?medicine . #Instances of that medication
        ?interactswith wdt:P2175 ?treats . #?treats is the medical condition treated by the drug that ?medicine interacts with.
        ?treats wdt:P31 wd:Q12135 . # where ?treats is an instance of mental disorder
  }
  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
}
```

New

```
SELECT DISTINCT ?medicine ?medicineLabel
  WHERE {
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
    {
        VALUES ?item {wd:DISEASE} # select all items connected to "DISEASE"
        ?item wdt:P2176 ?medicine . # medicine is the drug or therapy used to treat "DISEASE"
  }
}
```

Main differences:

- We have split up the queries fo fetching medication and for fetching the interactions. This simplifies the query and make it easier to deal with the data throughout later code
- We now make use of the property “drug or treatment used” instead of the ‘medical condition treated” property. Thi ensures that we don’t exclude some form of treatment which not directly a subclass of medication.

Interactions query Getting the Interactions

Old

```
SELECT DISTINCT ?medicine ?medicineLabel ?type ?typeLabel ?interactswithLabel ?treatsLabel
WHERE {
  {
      VALUES ?item {wd:Q12140} #selects all entries that are medicines
        ?medicine wdt:P279 ?item ; #?medicine is a subclass of those entries
             wdt:P2175 wd:Q181923 ; #which is selected only if the medical condition treated is Attention deficit Hyperactivity disorder
             wdt:P769 ?interactswith . #Significant drug interactions of that medicine
        ?type wdt:P31 ?medicine . #Instances of that medication
        ?interactswith wdt:P2175 ?treats . #?treats is the medical condition treated by the drug that ?medicine interacts with.
        ?treats wdt:P31 wd:Q12135 . # where ?treats is an instance of mental disorder
  }
  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
}
```

New

```
SELECT DISTINCT  ?interactswith ?interactswithLabel
  WHERE {
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
    {
        wd:MED wdt:P769 ?interactswith .  # the medications that interact with "MED"
    }
    UNION
    {
       ?interactswith wdt:P769 wd:MED .   # or the medications that the input medicine interacts with
    }
      ?interactswith wdt:P2175 ?treats .  # the disease which that medication treats
      ?treats wdt:P31 wd:Q112193867 ;     # if that disease is an instance of class of disease
                  wdt:P1995 wd:Q7867 .    # and the health specialty falls under psychiatry
      FILTER(?treats = wd:DISEASES )      # filters the results such that we only get the interactions
                                          # with a given medication if it treats the other selected disease
  }
```

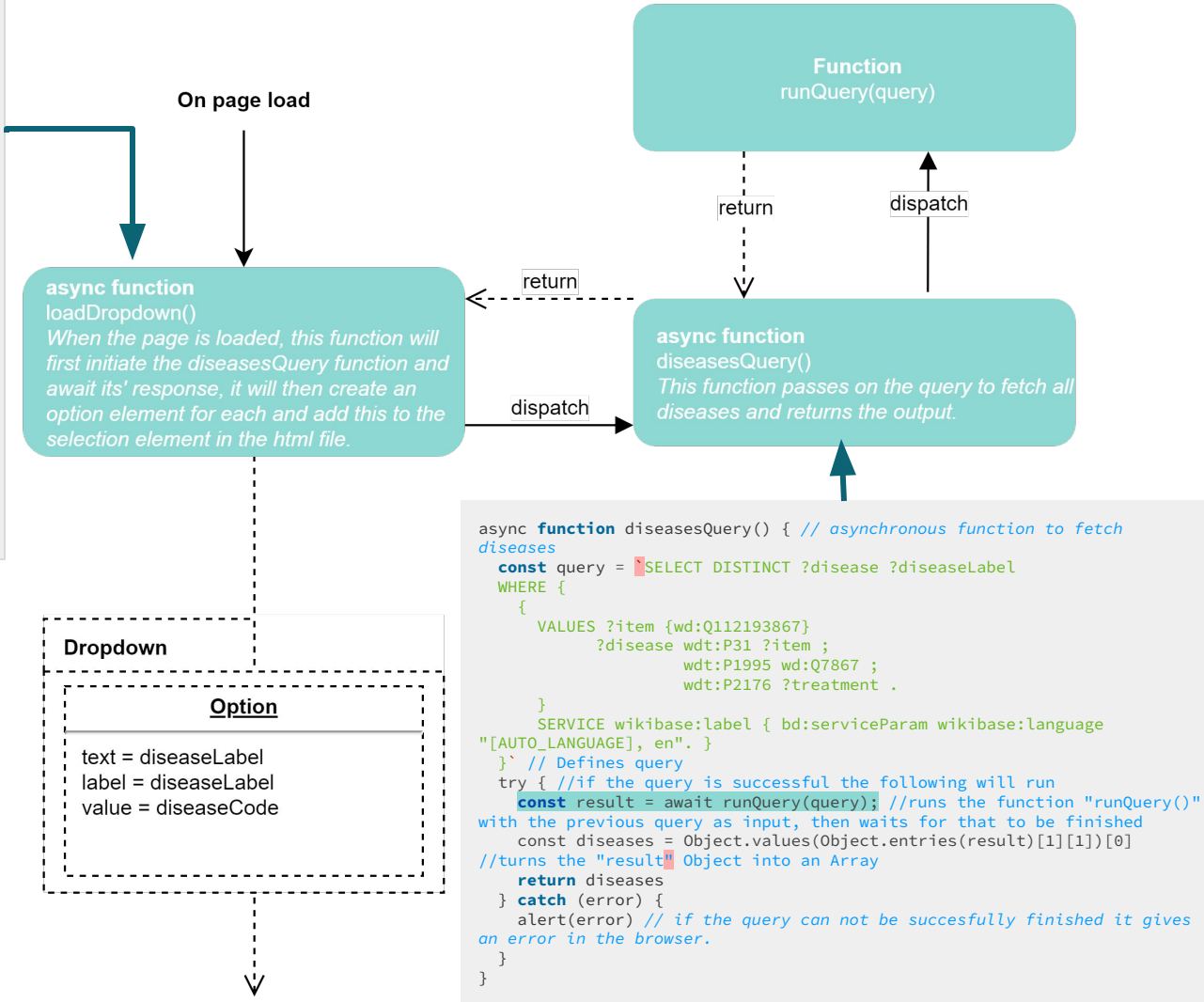
Main differences:

- We now consider and try to bypas potential inconsistencies for the “significant drug interactions’

Medication	**“Significant drug interactions”**
Methylphenidate	Tranylcypromine
	Isocarboxazid
	Phenelzine
	Procarbazine
	(±)-depenyl
tranylcypromine	Procarbazine

Loading the page

```python
async function loadDropdown() {
    const dropdown = document.getElementById("dropdown")
//tells javascript that whenever we refer to "dropdown"
    // we are refering to the html element with the id
"dropdown"
    const diseases = await diseasesQuery() //fetches the
diseases by running the function from the queries.js
    for (var i = 0, l = diseases.length; i < l; i++) {
        const option = document.createElement('option')
//for each element in the list
        // a new 'option' is created in the form
        option.text = diseases[i].diseaseLabel.value; //
attaches the name of the disease to the option
        option.label = diseases[i].diseaseLabel.value;
        option.value = diseases[i].disease.value;
//attaches the wikidata code to the option so it can
        //be passed on later
        option.classList.add()
        dropdown.add(option) // adds newly created option
to the <select> </select> element in the html
    }
    $(function ()
    { // function to alphabetically sort the diseases
        var select = $('#dropdown'); //selects the element
with the id reading "dropdown"
        select.html(select.find('option')
            .sort(function (x, y) {
                return $(x).text()
                    > $(y).text() ? 1 : -1;
            }))
    }
    )
}
```




```python
async function diseasesQuery() { // asynchronous function to fetch
diseases
  const query = `SELECT DISTINCT ?disease ?diseaseLabel
  WHERE {
    {
      VALUES ?item {wd:Q112193867}
            ?disease wdt:P31 ?item ;
                     wdt:P1995 wd:Q7867 ;
                     wdt:P2176 ?treatment .
      }
      SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE], en". }
  }` // Defines query
  try { //if the query is successful the following will run
    const result = await runQuery(query); //runs the function "runQuery()"
with the previous query as input, then waits for that to be finished
    const diseases = Object.values(Object.entries(result)[1][1])[0]
//turns the "result" Object into an Array
    return diseases
  } catch (error) {
    alert(error) // if the query can not be succesfully finished it gives
an error in the browser.
  }
}
```

HOME

Medchecker

Diseases

Asperger syndrome
Lewy body dementia
agoraphobia
alcohol abuse
amnesia
anorexia nervosa
antisocial personality disorder
anxiety disorder
attention deficit hyperactivity disorder
autism
bipolar I disorder
bipolar disorder
borderline personality disorder
bulimia
cocaine dependence

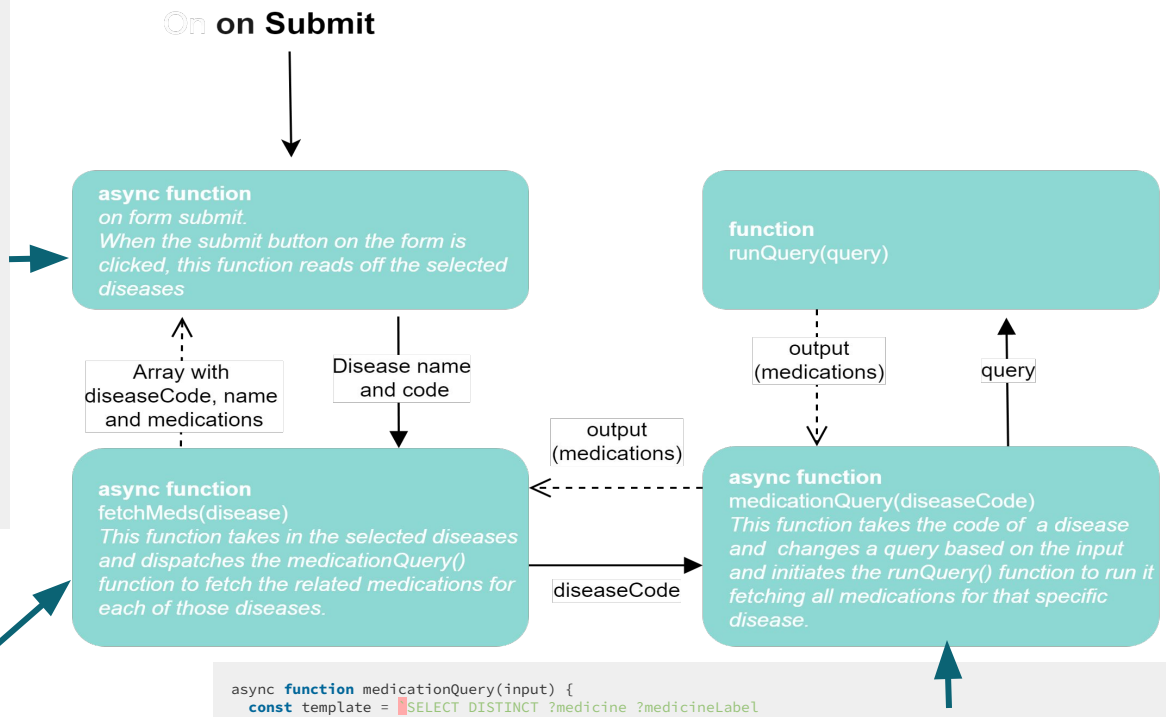
reset | submit

Your Selection:

Begin by selecting two diseases from the list. to select two hold ctrl while clicking with your mouse

Getting and manipulating the data

```
$("#submit").on("click", async function () { //adds event listener in the
button in the html file with the id "submit"
        $("#dropdown").attr("size", '5')
        var diseasesSelected = [] //clears the list before (in case the
function is ran multiple times)
        var diseasesSelected = $("select option:selected") // assigns all the
selected options to the variable selection
        var allMeds = []
        $('#diseaseList').empty() // clears the list element before running the
rest of the function (in case the function is ran multiple times)
        $('#div').empty()//clears the div before running the rest of the
function
        for (var i = 0, l = diseasesSelected.length; i < l; i++) { // loops
through all elements of the selection list
            const disease = document.createElement('li') //creates a list item
for each of the diseases selected by the user
                    disease.classList.add("w-full")
                    disease.innerHTML = diseasesSelected[i].label //allows us
to display the name of the disease
            $('#diseaseList').prepend(disease) // adds the list element to the
list
            const meds = await fetchMeds(diseasesSelected[i])// passes on the
selected diseases and initiates the query to fetch their medicines
            allMeds.push(meds)
        }
    fetchInteractions(allMeds)
    })
```




```
async function fetchMeds(selection) {
    const diseaseCode = selection.value.slice(31) // leaves only the wikidata
code of that object
    const outputMedicationQuery = await medicationQuery(diseaseCode)

    var medication = [] //clears out the list of medication before the
function is ran (incase is is ran multiple times)
    var medicationList = []

    for (var j = 0, m = outputMedicationQuery.length; j < m; j++)
    {

medicationList.push([outputMedicationQuery[j].medicine.value.slice(31),
outputMedicationQuery[j].medicineLabel.value]) //adds the medication to the
list, once again the slice makes sure we only get the item code
    }
    medication.push([diseaseCode, selection.label, medicationList]) //adds
the code and name of a disease and all the medications that treat it
    return medication
}
```

```
async function medicationQuery(input) {
  const template = `SELECT DISTINCT ?medicine ?medicineLabel
  WHERE {
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en". }
    {
        VALUES ?item {wd:DISEASE}
        ?item wdt:P2176 ?medicine;
  }
}`
const querymeds = template.replace('DISEASE', input) //we replace the string "variable" in the query by the
code of the disease, stored in "input"
try { //if the query is successful the following will run
  const result = await runQuery(querymeds) // first waits for the result to be fetched, otherwise we will
not catch the promise
  const medication = Object.values(Object.entries(result)[1][1])[0] //turns the "result" Object into an
Array
  return medication //we need this so that when we run the function elsewhere we can store the results
} catch (error) {
    alert(error) // if the query can not be successfully finished it gives an error in the browser.
  }
}
```

```
async function fetchInteractions(selected) {
         const disA = (selected[0][0])[0]
        const medsA = (selected[0][0])[2]
        const disB = (selected[1][0])[0]
        const medsB = (selected[1][0])[2]
        const medsAllInclInteractions = []
    for (var i = 0, l = medsA.length; i < l;
i++){
        const interactions = await
interactionsQuery((medsA[i])[0], disB)
        medsAllInclInteractions.push({
                diseaseName:
(selected[0][0])[1],
                diseaseCode: disA,
                medicationName: (medsA[i])[1],
                medicationCode: (medsA[i])[0],
                interactswith: interactions
            })}
    for (var j = 0, m = medsB.length; j < m;
j++){
        const interactions = await
interactionsQuery((medsB[j])[0], disA)
        medsAllInclInteractions.push({
                diseaseName:
(selected[1][0])[1],
                diseaseCode: disB,
                medicationName: (medsB[j])[1],
                medicationCode: (medsB[j])[0],
                interactswith: interactions
            })}
    makeMatrix(medsAllInclInteractions)
}
```

after all medications have been fetched

function
runQuery(query)

output (medications)

query

async function
fetchInteractions(disease, medications)
This function takes in an array with the names and codes of the diseases and their medications. Then it will run the interactiondQuery() function and produce an object for each medication using its output.

diseaseCodeA and medicationCode B

diseaseCodeB and medicationCodeA

output (medications)

async function
interactionsQuery(diseaseCode, medicationCode)
This function takes the code of a disease and medication and changes a query based on the input and initiates the runQuery() function to run it fetching all interactions for that specific medication, if it interacts with a medication treating that specific disease .

Object:object

diseaseName = disease.name
diseaseCode = disease.code
medicationName = medications.name
medicationCode = medication.code
interactsWith = output

makeMatrix()

```
async function interactionsQuery(medicine, disease) {
  const querytemplate = `SELECT DISTINCT  ?interactswith ?interactswithLabel
  WHERE {
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE], en".
{
      wd:MED wdt:P769 ?interactswith .
    } UNION {
       ?interactswith wdt:P769 wd:MED .    }
      ?interactswith wdt:P2175 ?treats .
      ?treats wdt:P31 wd:Q112193867 ;
                  wdt:P1995 wd:Q7867 .
      FILTER(?treats = wd:DISEASES )}`
  const queryint = querytemplate.replaceAll("MED", medicine).replace("DISEASES",
disease)
  try { //if the query is successful the following will run
    const result = await runQuery(queryint)
    const medication = Object.values(Object.entries(result)[1][1])[0] //turns the
"result" Object into an Array
    return medication
  } catch (error) {
      alert(error) // if the query can not be succesfully finished it gives an error
in the browser.
    }}
```

(index)	diseaseName	diseaseCode	medicationName	medicationCode	interactswith
0	'attention deficit hyperactivity disorder'	'Q181923'	'(±)-deprenyl'	'Q402633'	Array(1)['venflafaxine']
1	'autism'	'Q38404'	'venlafaxine'	'Q898407'	Array(1)['(±)-deprenyl']

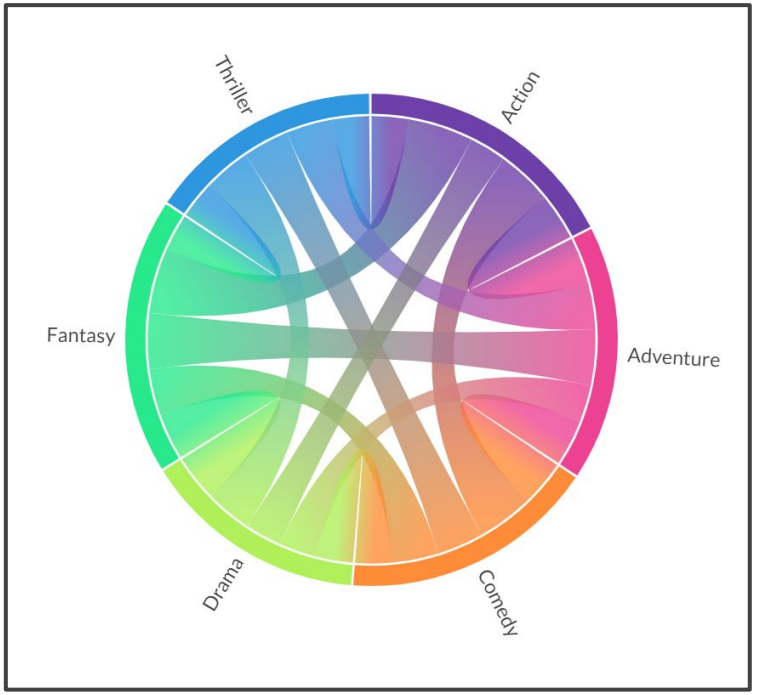
```
function makeMatrix(allMeds) {
    let filtered = allMeds.filter(med => med.interactswith.map(x =>
x.interactswithLabel.value).length>0)
    if (filtered.length == 0) {
        alert("No interactions")
    }
    const size = filtered.length
    const matrix = []
    filtered.forEach((med) => {
        let row = Array(size).fill(0)
        for (var i = 0, l = med.interactswith.length; i < l; i++){
                let index = filtered.map(x =>
x.medicationName).indexOf((med.interactswith[i].interactswithLabel.value))
                row.splice(index, 1, 1)
        }
        matrix.push(row)
    })
    dataVisualization(filtered, matrix)
}
```

```
[]

[[00]]

[[01]]

[[01]
 [00]]

[[01]
 [10]]
```

Visualization

The idea of visualization

	Action	Adventure	Comedy	Drama	Fantasy	Thriller
Action	0	5	6	4	7	4
Adventure	5	0	5	4	6	5
Comedy	6	5	0	4	5	5
Drama	4	4	4	0	5	5
Fantasy	7	6	5	5	0	4
Thriller	4	5	5	5	4	0



[8]

How it works?

	Action	Adventure	Comedy	Drama	Fantasy	Thriller
Action	0	5	6	4	7	4
Adventure	5	0	5	4	6	5
Comedy	6	5	0	4	5	5
Drama	4	4	4	0	5	5
Fantasy	7	6	5	5	0	4
Thriller	4	5	5	5	4	0

```python
matrix = [
    [0, 5, 6, 4, 7, 4],
    [5, 0, 5, 4, 6, 5],
    [6, 5, 0, 4, 5, 5],
    [4, 4, 4, 0, 5, 5],
    [7, 6, 5, 5, 0, 4],
    [4, 5, 5, 5, 4, 0],
]

names = ["Action", "Adventure", "Comedy", "Drama", "Fantasy", "Thriller"]
```

Preparatory work

```
_____________________________________Set-Up_____________________________

ariable ideas taken from: https://stackoverflow.com/questions/43

 const names = []
 const colors = []
 const interactions = matrix

 const opacityDefault = 0.8
 const margin = { left: 90, top: 90, right: 90, bottom: 90 },
  width = 1000,
  height = 1000,
  innerRadius = 1000 * .30,
  outerRadius = innerRadius * 1.1;
```

```
const chord = d3.chord()
    .padAngle(0.15)
    .sortSubgroups(d3.descending)

//moved color variable to all variables
const color = d3.scaleOrdinal()
    .domain(d3.range(names.length))
    .range(colors)

const arc = d3.arc()
    .innerRadius(innerRadius)
    .outerRadius(outerRadius)

const ribbon = d3.ribbon()
    .radius(innerRadius)

const tooltip = d3.select("#chart")
    .append("div")
    .style("opacity", 0)
    .attr("class", "tooltip")
    .style("background-color", "white")
    .style("border", "solid")
    .style("border-width", "1px")
    .style("border-radius", "5px")
    .style("padding", "10px")
```

Creating SVG

```
Create SVG

const svg = d3.select("#chart")
    .append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate(" + (width / 2 + margin.left)
        + "," + (height / 2 + margin.top) + ")")
    .datum(chord(interactions));
```

```
const chord = d3.chord()
    .padAngle(0.15)
    .sortSubgroups(d3.descending)
```

chords

chords.groups

Drawing inner arcs

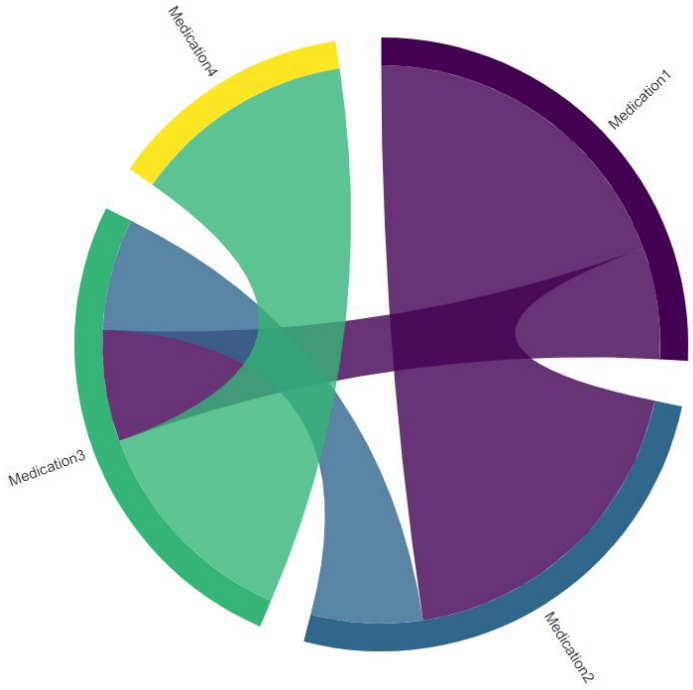
```
//Make a group for every "groups" dataset
const innerArcs = svg.selectAll("g.group")
    .data(function(chords) { return chords.groups; })
    .enter().append("g")
    .attr("class", "group")

//Visualize arcs using "path"
innerArcs.append("path")
    .style("fill", function(d) { return color(d.index); })
    .attr("d", arc);// missing radius bounds
```

```
const arc = d3.arc()
    .innerRadius(innerRadius)
    .outerRadius(outerRadius)
```

Drawing interactions (chords)

```
svg.append("g")
  .attr("class", "chords")
  .selectAll("path")
  .data(function(chords) { return chords; })
  .enter().append("path")
  .style("fill", function(d) { return color(d.source.index); })
  .style("opacity", opacityDefault)
  .attr("d", ribbon)
```

```
const ribbon = d3.ribbon()
    .radius(innerRadius)
```



Adding outer arcs

```
//make a list of disease names and medication groups
const groups = d3.groups(data, d => d.diseaseName)
const chordData = chord(interactions).groups
```

```
for (var i = 0; i < groups.length; i++) {
/*every start and end index represent the indexes of
first and last medication for every disease arc*/
  const sIndex = data.map(x => x.medicationName)
    .indexOf((groups[i])[1][0].medicationName)
  const eIndex = data.map(x => x.medicationName)
    .indexOf((groups[i])[1][(groups[i][1].length)-1].medicationName)
```

```
//Parameters for an outer arc
var diseaseGroup = groups[i]
var outerArc = d3.arc()
  .innerRadius(innerRadius + 160)
  .outerRadius(outerRadius + 100)
  .startAngle(chordData[sIndex].startAngle)
  .endAngle(chordData[eIndex].endAngle)
```

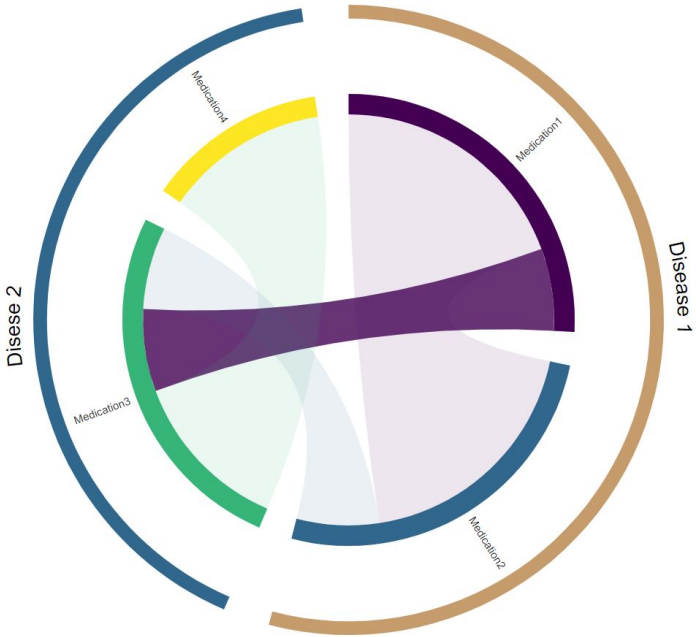
```
//Drawing outer arc
svg.append("path")
  .style('fill', colors[i])
  .attr("class", "superGroup")
  .style("opacity", "50%")
  .attr("id", `outerGroup${i}`)
  .attr("d", outerArc);
```

Interactions

Interactions 1 - fading of chords

```
//The Mouseover event
.on('mouseover', function (event, d, i) {
  return svg.selectAll("g.chords path")
    .filter(function(chordData) {
      return chordData.source.index != d.source.index
        || chordData.target.index != d.target.index;
    })
    .transition()
    .style("opacity", 0.1);
})
```

```
//The mouseout event
.on('mouseout', function () {
  return svg.selectAll("g.chords path")
    .transition()
    .style("opacity", opacityDefault);
})
```



Interactions 2 - tooltip box

```
//The Mouseover event
.on('mouseover', function (event, d, i) {
  //Puting text into the tooltip
  tooltip
    .html("Source: " + names[d.source.index]
      + "<br>Target: " + names[d.target.index])
    .transition()
      .duration(400)
      .style("font-size","16px");
```

```
//The mouseout event
.on('mouseout', function () {
  //added a nice transition that would
  tooltip
    .transition()
    .duration(400)
    .style("font-size","0px")
```

Discussion

Limitations

-Results determined by accuracy & completeness of the medication/interactions list on wikidata

-(Currently) only for mental health illnesses + medications

-Only 2 diseases at a time are checked

-Long result loading time

Advantages

-Reassurance and comfort, patient won't go to the GP for everything → less stress on the GP/healthcare system

-Reassurance and comfort → less stress on the healthcaresystem/GPs due to reduced trips

Overview problems and solutions

	Problem	Solution
Week 2	*Accessing and displaying data resulting from a Query.*	*The query returns an object which we can turn into an array using Object.values*
Week 3	*For repeated selection, diseases accumulated in list displays.*	*Clear each list and relevant container before repeating the function.*
	The value of a disease or medicine is an URL, not the code displayed on wikidata.	*That first part is the same for all wikidata entries, we use the slice function to remove everything but that code.*
Week 4	Filling in an arbitrary variable into a query from Javascript.	*create a query template with some text (wd:DISEASE or wd:MEDICINE), then use template.replaceAll('text', input) where input is the code of the medicine or disease.*
	Relations in wikidata are not bidirectional	*Query of interactions is a UNION of results of medications which have a significant drug interaction with medA and medications where medA is the target of the significant drug interactions*
	Data for diagram must be in the form of a matrix.	*We filter out the non relevant entries (those with no interactions) the dimensions of the matrix will be equal to the lentgh of this list. We then create an array for each disease, fill it wilth all zeroes, find the index of each of its interactions within this list and replace those indexes with ones in the array*
Week 5	When generating the outer arcs we needed to find a way to have the code check for itself where the start and end of a group was	*We have d3.js group the medications by disease, we then tell the program to, within the big list of all medications, find the index of the first and last entry of that group.*
	We need to generate enough colours to be able to assign one to each medication	*Wrote a function that takes in the length of the list of medications and randomly generates that number of colour codes.*

Conclusion

Goals achieved

-Easy, efficient platform achieved

-Accesses data from wikidata via SPARQL and implements it using javascript

-Nice visualization achieved through javascript

-Website made using html & css (tailwind)

Future outlook

-Implementation of more illness categories besides mental health

-Comparison of more than 2 diseases

-Implementation of more databases

References

[1] Center for Drug Evaluation and Research. (2017, October 27). CDER conversation: Evaluating the risk of Drug-Drug Interactions. U.S. Food and Drug Administration. Retrieved November 4, 2022, from https://www.fda.gov/drugs/news-events-human-drugs/cder-conversation-evaluating-risk-drug-drug-interactions

[2] Rao ME and Rao DM (2021) The Mental Health of High School Students During the COVID-19 Pandemic. Front. Educ. 6:719539. doi: 10.3389/feduc.2021.719539

[3] Q181923. (2022, November 23). Wikidata. Retrieved 12:00, December 2, 2022 from https://www.wikidata.org/w/index.php?title=Q181923&oldid=1777715802.

[4] Aarts, S. (2022, November 18). Queries using SPARQL. Retrieved December 12, 2022 from https://docs.google.com/presentation/d/1QRDEqBotlQh563yLR2aYQkCqB5Jf8a2dZh55GZmZKGg/.

[5] Aarts, S, (2022, December 2). Dropdown.js. Retrieved December 2, 2022 from https://github.com/Snrts/Project_Repo/blob/main/scripts/dropdown.js.

[6] Cambridge Semantics, Inc. (n.d.). SPARQL Query Types and Clauses. SPARQL query types and clauses. Retrieved December 5, 2022, from https://docs.cambridgesemantics.com/anzograph/v2.5/userdoc/sparql-queries.htm

[7] Cambridge Semantics, Inc. (n.d.). WHERE clause. Where clause. Retrieved December 5, 2022, from https://docs.cambridgesemantics.com/anzograph/v2.5/userdoc/where.htm

[8] Chord | PlotAPI Docs. (n.d.). https://plotapi.com/docs/visualizations/chord/