Relation visualizer branch description

Dette er er stort pull request, så jeg prøver lige at beskrive ændringerne fra toppen

Database

Databasen bruger nu intervaller på formen ",x", "x,y" og "x+" for at undgå tvetydighed hvis der en dag skulle komme negative tal.

risk_ratios.py

Før brugte vi almindelige data frames for risk ratios, men nu har jeg lavet en decideret klasse `RiskRatioTable`, der nedarver `data_frame`. Udover de sædvanlige `data_frame`-parametre indeholder den nu også

- Tuningparameteren for en eventuel interpolation, `lambd`
- `bounding_method` som kommunikerer hvordan interpolationsfunktionen skal opføre sig på uendelige intervaller.
- `variances` som er en liste af samme længde som risk ratio-tabellen der indeholder en varians for alle værdier (eller None, hvis den ikke er angivet).

interpolation.py

Nu er interpolationsfunktionen lineær og ikke kubisk endelige intervaller. I `interpolation.py` betyder det at inputtet til `spline_system` kun er de endelige intervaller. For at håndtere varianser på riskratio-værdierne bliver de aflæst i `collect_interpolated_data_frame_spline`. Hvis varianserne er 0 bliver de tilsvarende indgange i interpolationsprediktionsmatricen taget ud i en separat `X_fixed` for at kunne lave Lagrange multiplier metode.

spline.py

Hovedfunktionen `get_maximum_likelihood_estimate` tager nu scales, som er varianser på RR-værdierne og hvis en varians er 0 laver den i stedet lagrange multiplier for at indsætte den pågældende RR-værdi som en constraint.

interpolation_tails.py

Denne klasse er indført for at indsætte bounds på bounds-objektet i `interpolationTable`-objektet. Den kan kun lave `minbounded of_riskratio_values`og `maxbounded of_riskratio_values` - minbounded er altid på fordi vi vil ikke ekstrapolere en sygdom fuldstændig væk. `of_riskratio_values` referer til at begrænsningen laves af riskratiotabellen, i modsætning til af interpolationstabellen.

caller.py og Relations.json

Dette modul udregner nu også `Relations.json-filen`, som indeholder relationsgrafen. Den har struktur: `{nodeName: {type:t, color:c, ancestors:[nodeName1, ..., nodeNameN]}}` hvor `ancestors` er en liste af alle de nodes der er nødvendige at udregne inden man udregner den pågældende node. Typen `t` er enten

1. **Input factor**, hvilket vil sige at det er et spørgsmål, som bliver stillet til brugeren. `caller.py` aflæser dette fra `FactorDatabase.json`-filen.
2. **Computed factor**, hvilket er en deterministisk funktion af input factors. `caller.py` aflæser disse nodes og deres ancestors fra den hard-codede fil `death-causes-app/public/ComputedFactorsRelations.json`
3. **Condition**, hvilket vil sige at man ikke dør af den, men at vi alligevel estimerer sandsynligheden for at have den condition hvert år. Der er ikke lavet yderligere funktionalitet for dette led, så den indgår ikke rigtigt endnu.
4. **Death cause category**, hvilket er en forening af flere death causes. Vi har dem med som en klasse for at vi kan knytte risk ratio tables til death cause categories i stedet for kun death causes. `caller.py` aflæser disse nodes fra mappestrukturen i `Database`-mappen. Eventuelle faktor-ancestors til death cause categories aflæses i de indlæste risk-ratiotabeller.
5. **Death cause**, hvilket er end-notes der hver har deres egen sandsynlighed for at medføre dødsfald. `caller.py` udregner ancestors på samme måde som for Death cause categories.

og farven `c` er en hex color. Lige nu får hver node en farve ved at transformere navnet om til en farve med funktionen `make_hex_color` fra `generate_color.py`.

Relationlink.ts

Dette modul indlæser `Relations.json` filen og pre-udregner en del dictionaries, der gør senere udregning hurtigere. Af vigtige funktioner er

- `makePlottingInstructions` som med udgangspunkt i en enkelt node udregner en graf over alle nodes der er direkte forfædre eller efterkommere af den pågældende node.
- `getSuperDescendantCount` giver antallet af risk ratiofiler hvor en enkelt node har efterkommere. Det bruges til at lave rækkefølgen af input factors.
- `sortedNodes` er et objekt der angiver rækkefølgen på nodes sådan at afhængige nodes kommer efter de nodes, de er afhængige af. Fordi vi har 5 klare kategorier (Input factor, Computed factor, Condition, Death cause category og death cause) er sortedNodes også delt op i 5. Rækkefølgen udregnes af `KahnSort.js`

FactorDatabase.json

Denne håndskrevne fil indeholder alle input factors, deres spørgsmål, og argumenterne

- `type` angiver om det er en numerisk eller string-spørgsmål
- `longExplanation` er det spørgsmål brugeren skal stilles uden spørgsmålstegn og enheder.
- `placeholder` er enheden. For string-spørgsmål kan det også være en kommando (e.g. Choose Race). Den hedder placeholder fordi det er den tekst, der står i forminputfeltet.
- `requiredDomain`. Hvis ikke brugeren angiver en tom værdi eller en værdi inden for dette interval kaster programmet en error.
- `recommendDomain`. Som foregående, blot med warning.
- `initialValue` den værdi som antages først. For string-spørgsmål er dette altid det samme som placeholder for der er ikke rigtigt en placeholder for form select. For numeriske variable default'er den bare til en tom string.
- `options` for string-spørgsmål er dette listen af svarmuligheder.
- `units` for numeriske spørgsmål er dette en dictionary over hvilke skalafaktorer der er mellem den rigtige enhed(indikeret med `placeholder`) og de alternative enheder.

- `helpJson` en string der bliver omdannet til html med markdown inde i help-boksen
- `derivables`. Hvis svaret på et spørgsmål kan udledes ved hjælp af nogen af svarene på et andet string-spørgsmål bør det indikeres her, for at brugeren ikke stilles unødvendige spørgsmål. Den har formen `{causativeFactor1:{causalFactorLevel1: causedFactorLevel1,...},...}`, hvor meningen er at denne input factor fx skal have værdien `causedFactorLevel1` hvis `causativeFactor1` har værdien `causalFactorLevel1`.

App.tsx

Objektet `RelationLinkData` indlæses nu som state i `App.tsx`, da både `VizWindow.tsx` og `QuestionMenu.tsx` skal bruge det. Førstnævnte bruger det til at lave relation visualizer grafen samt farvelægge `BarChart.tsx` mens sidstnævne bruger det til at udregne rækkefølge af input factor spørgsmål og lave links til relation visualizergrafer.

Derudover er der der kommet variablen `elementInFocus` som sætter en bestemt node i `relationLinkData` i fokus. Lige nu bruges du kun til at vælge hvilken node i `relationLinkViz` der er øverst.

QuestionMenu.tsx

Denne component har fået nye tilstande der har stor betydning for layout'et

1. Man starter `answerProgress==ANSWERING` og `view==QUESTION_MANAGER`. Her vises der et kort med spørgsmål og en videre- og tilbageknap.

2. Dernæst havner man i `answerProgress==FINISHED` og `view==QUESTION_MANAGER`. Her vises der det samme kort, men med en besked om at man har svaret på alle spørgsmål. Der er så en mulighed for at gå videre til næste tilstand;
3. Her er `answerProgress==FINISHED` og `view==QUESTION_LIST`. Her vises der en liste med Input factorerne i forkortet version, da man allerede har set det i foregående tilstand.

Af andre nye states er

- `factorAnswerScales`. Når man ændrer en unit bliver den nye værdi gemt heri. Som standard er dette objekt tomt. Når en faktor ikke er i dette objekt betyder det at vi blot bruger standard enheden i de videre udregninger. Dette state ændres af `handleUnitChange`.
- `hasBeenAnswered`. Dette er en liste over alle de spørgsmål som er blevet besvaret. Efter at have ændret funktion flere gange i processen, bliver det nu kun brugt til at afgøre hvilke input factorer som der laves check for non-ignored missing.
- `activelyIgnored` er en dictionary over hvilke indgange der er ignored af brugeren. Den sættes med `handleIgnoreFactor`.
- `currentFactor` når vi er i `view==QUESTION_MANAGER` er dette factornavnet på den factor som er inde i question manager kortet.
- `windowWidth` er den overordnede bredde på vinduet som `react-bootstrap` også bruger til at afgøre om skærmen er `xs`,`lg` osv. Vi bruger den til at afgøre om helpboksens placering skal være til venstre eller højre. Den opdateres ved resize.
- `factorMaskings`. Hvis en input factor er defineret til at være *derivable* af en anden stat i `FactorDatabase.json` og det faktisk er den værdi, som den anden factor har, sætter vi "overskrivningen" ind i `factorMaskings`. Det vil sige at `factorAnswers` egentlig peger på den gamle uofficielle værdi og i `handleSubmit` overskrives den uofficielle værdi i `factorAnswersSubmitted`.
- `changedSinceLastCommit` er en boolean variabel som angiver om factorAnswers har ændret sig siden sidst. Det bruges til at angive om videre/recompute-knappen skal indikere, at den gerne vil trykkes på.

Derudover er der introduceret den nye klassevariabel `factorOrder` som udregnes af `Factors.ts`-klassen ved hjælp af `RelationLinkData`-objektet. Den angiver hvilken rækkefølge input factorerne skal stilles i.