NIvis

2022-10-14

Contents

Chapter 1

Introduction

Chapter 2

Downloading and prepaering example data

Chapter 3

Time series

3.1 Raw data

3.2 Scaled data

Chapter 4

Maps

4.1 Raw data

4.2 Scaled data

4.2.1 Jerv

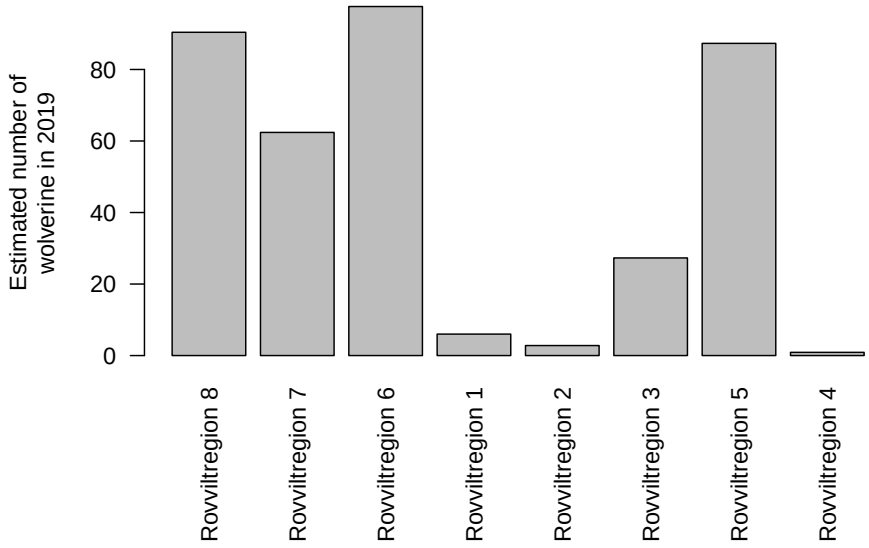
4.2.1.1 Prepare NI data

The jerv (wolverine) data was downloaded using the `R/singleIndicator.R` script and the importDatasetApi() function„ and subsequently the assembleNiObject() function, so now I can simply import it.

```
jerv <- readRDS("data/jerv_assemble.rds")
```

This data file contains the raw data in the form of expected values for each BSunits (municipalities). But we actually want to keep the original geometeris of the eight rovviltregioner, and so we need to focus in the ICunits instead.

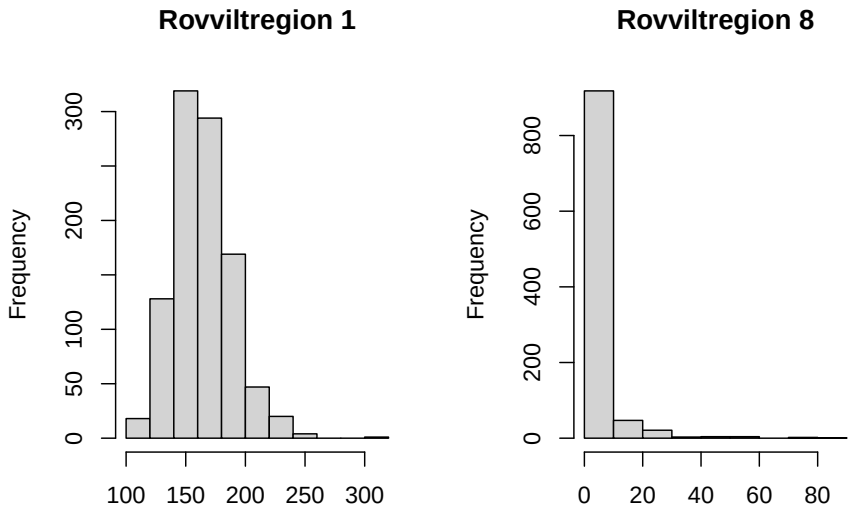
```
par(mar=c(9,5,1,1))
barplot(jerv$indicatorValues$`2019`$expectedValue,
        names.arg = jerv$indicatorValues$`2019`$ICunitName,
        las=2,
        ylab = "Estimated number of\nwolverine in 2019")
```



The data also contains upper and lower quantiles, but we can also get the full probability distribution and sample from it to get standard deviations. but also as probability functions that we can sample from:

```
# bruker tradOb siden custumDist er NA. Dette er ikke en generisk løsning.
obstype <- rep("tradObs", nrow(jerv$indicatorValues$'2019'))

#myYears <- as.character(c(1990,2000,2010,2014,2019))
myYears <- as.character(c(2019))

for(i in 1:length(myYears)){
# print(i)

myMat <- NIcalc::sampleObsMat(
  ICunitId           = jerv$indicatorValues[[i]]$ICunitId,
  value              = jerv$indicatorValues[[i]]$expectedValue,
  distrib            = jerv$indicatorValues[[i]]$distributionFamilyName,
  mu                 = jerv$indicatorValues[[i]]$distParameter1,
  sig                = jerv$indicatorValues[[i]]$distParameter2,
  customDistribution = jerv$indicatorValues[[i]]$customDistribution,
          obsType = obstype,
          nsim = 1000

)
assign(paste0("myMat", myYears[i]), myMat)
```

```
}
#> Warning: replacing previous import 'distr::plot' by
#> 'graphics::plot' when loading 'NIcalc'

par(mfrow = c(1,2))
hist(myMat2019[1,], main = "Rovviltregion 1", xlab = "")
hist(myMat2019[8,], main = "Rovviltregion 8", xlab = "")
```

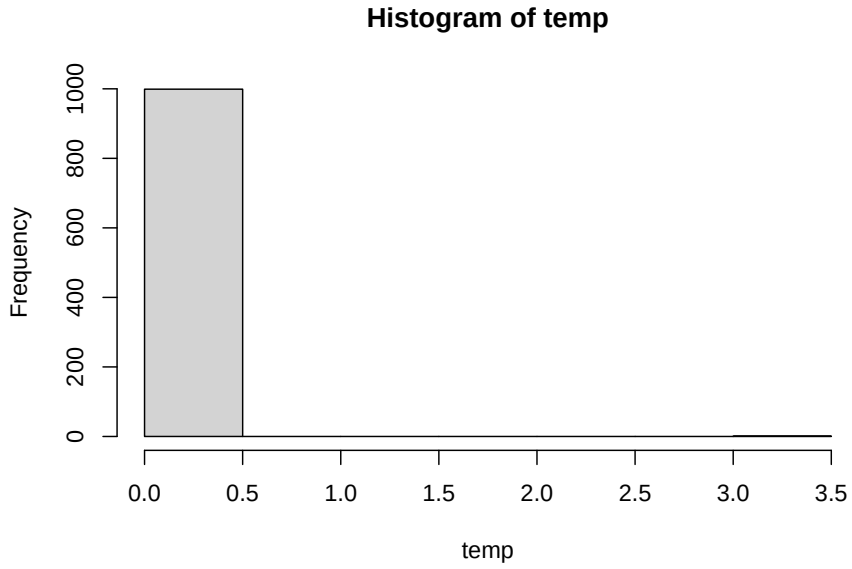


For some reason the extected values are far from the mean of these distributions. I did this exercise once before, and did not get this problem then. I think the difference is that I use eco = NULL this time, in the `importDatasetApi()`, and this cause the output to somehow split into forest and alpine ecosystems. I will ignore this here for this example.

I can also get the reference values in the same way, and then divide one by the other to get scaled values

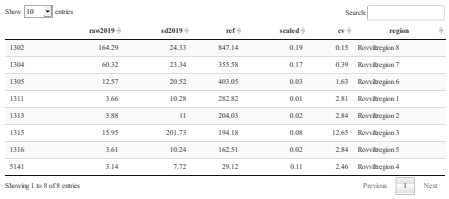
```
myMatr <- NIcalc::sampleObsMat(
            jerv$referenceValues$ICunitId,
            jerv$referenceValues$expectedValue,
            jerv$referenceValues$distributionFamilyName,
            mu = jerv$referenceValues$distParameter1,
            sig = jerv$referenceValues$distParameter2,
            customDistribution = jerv$referenceValues$customDistribution,
            obsType = obstype,
```

```
            nsim =1000
        )

temp <- colSums(myMat2019)/colSums(myMatr)
hist(temp)
```



Then I will create a data frame with the mean indicator values and the SD.

```
library(matrixStats)
#> Warning: package 'matrixStats' was built under R version
#> 4.1.3
jerv_tbl <- data.frame("raw2019" = round(rowMeans(myMat2019), 2),
                       "sd2019"  = round(matrixStats::rowSds(myMat2019), 2),
                       "ref"     = round(rowMeans(myMatr), 2))
jerv_tbl$scaled <- round(jerv_tbl$raw2019/jerv_tbl$ref, 2)
jerv_tbl$cv <- round(jerv_tbl$sd2019/jerv_tbl$raw2019, 2)
jerv_tbl$region <- jerv$indicatorValues$`2019`$ICunitName
DT::datatable(jerv_tbl)
```



Show 10 entries Search:

	raw2019	sd2019	ref	scaled	cv	region
1302	164.29	24.33	847.14	0.19	0.15	Rsvvilregion 8
1304	60.32	23.34	355.58	0.17	0.39	Rsvvilregion 7
1305	12.57	20.52	403.05	0.03	1.63	Rsvvilregion 6
1311	3.66	10.28	282.82	0.01	2.81	Rsvvilregion 1
1313	3.88	11	204.03	0.02	2.84	Rsvvilregion 2
1315	15.95	201.73	194.18	0.08	12.65	Rsvvilregion 3
1316	3.61	10.24	162.51	0.02	2.84	Rsvvilregion 5
5141	3.14	7.72	29.12	0.11	2.46	Rsvvilregion 4

Showing 1 to 8 of 8 entries Previous 1 Next

This is a special case maybe, because the sd is often larger than the mean.

Btw, we could use inbuilt NIcalc functions to get the indicator value, like I do below, but that will aggregate to regions, and we want to keep the original geometry.

```
jervComp <- NIcalc::calculateIndex(
  x       = jerv,
  nsim     = 1000,
  awBSunit = "terrestrialArea",
  fids     = F,     # should fidelities be ignored in
                    # the calculation of Wi?
  tgroups  = F, # should grouping of indicators
                    # into trophic and key indicator
                    # groups be ignored
  keys     = "specialWeight", #"ignore",
)
#> Indices for NIunits 'wholeArea', 'E', 'S', 'W', 'C', 'N'
#> and years '1990', '2000', '2010', '2014', '2019' will be calculated.
#> The 30 index distributions will each be based on  1000 simulations.
#> There are 8 ICunits with observations in data set 'jerv'.
#>
#> Calculating weights that are the same for all years .....
#>
#> Sampling reference values .....
#>
#> Sampling and scaling indicator observations from  1990 .....
#>
#> Sampling and scaling indicator observations from  2000 .....
```

```
#>
#> Sampling and scaling indicator observations from  2010 .....
#>
#> Sampling and scaling indicator observations from  2014 .....
#>
#> Sampling and scaling indicator observations from  2019 .....
plot(jervComp$wholeArea)
```

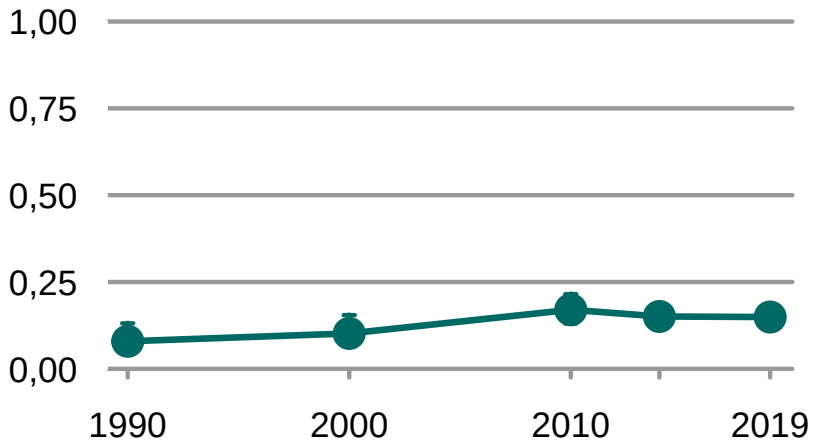


Figure 4.1: The scaled indicator values for wolverine across Norway.

4.2.1.2 Get geometries

Then I can get the spatial geometries associated with the data. There are the so called rovviltregioner. There are eight of them. They are actually linked to the BS-units (municipalites), but we don't want to plot the outlines of the municipalities.

Add text about how we got the json file and converted it to a shape file

```
path <- "P:/41201612_naturindeks_2021_2023_database_og_innsynslosning/Pilot_Forbedring
```

```
library(sf)
#> Linking to GEOS 3.9.1, GDAL 3.2.1, PROJ 7.2.1; sf_use_s2() is TRUE
rov <- sf::read_sf(path)
rov <- sf::st_make_valid(rov)
rov <- rov[rov$area!="DEF jerv",]
```

Clip it against the outline of Norway to make it look more pretty

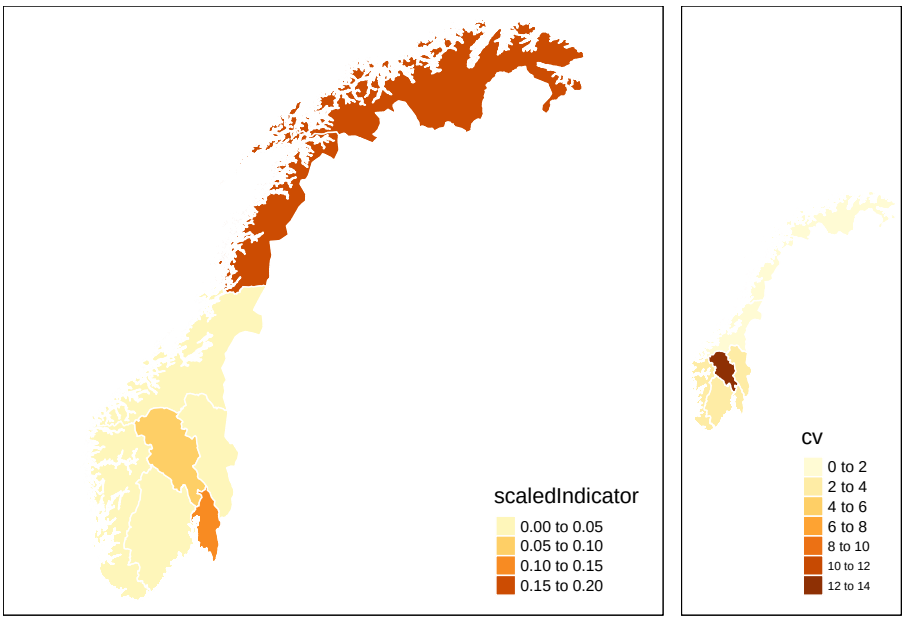
```
path <- "data/outlineOfNorway_EPSG25833.shp"
nor <- sf::read_sf(path)
nor <- st_transform(nor, crs=st_crs(rov))
```

```
rov <- st_intersection(rov, nor)
#> Warning: attribute variables are assumed to be spatially
#> constant throughout all geometries
```

4.2.1.3 Link data and geometries

```
rov$scaledIndicator <- jerv_tbl$scaled[match(rov$area, jerv_tbl$region)]
rov$cv <- jerv_tbl$cv[match(rov$area, jerv_tbl$region)]
rov$raw <- jerv_tbl$raw2019[match(rov$area, jerv_tbl$region)]
```

```
library(tmap)
one <- tm_shape(rov)+
  tm_polygons(col="scaledIndicator",
              border.col = "white")

two <- tm_shape(rov)+
  tm_polygons(col="cv",
              border.col = "white")

three <- tm_shape(rov)+
  tm_polygons(col="raw",
              border.col = "white")


tmap_arrange(one, two,
             widths = c(.75, .25),
             heights = c(1, 0.5))
#> Some legend labels were too wide. These labels have been resized to 0.61, 0.53, 0.53. Increase
```


scaledIndicator
0.00 to 0.05
0.05 to 0.10
0.10 to 0.15
0.15 to 0.20
cv
0 to 2
2 to 4
4 to 6
6 to 8
8 to 10
10 to 12
12 to 14

Chapter 5

Other figures