NIvis

2022-10-18

Contents

1 Introduction **5**

2 Downloading and preparing example data **7**

3 Time series **13**
3.1 Raw data 13
3.2 Scaled data 14

4 Maps **19**
4.1 Raw data 19
4.2 Scaled data 19

5 Other figures **27**
5.1 Gradient density plots for interactive maps 27
5.2 Ecosystem fidelity 37
5.3 Impact factor wordclouds 46
5.4 Data type 51

Chapter 1

Introduction

Chapter 2

Downloading and preparing example data

```
if(!require(NIcalc)){
  devtools::install_github("NINAnor/NIcalc", build_vignettes = F)
}
library(NIcalc)
```

Fill in your username (NINA email) and password.

```
myUser <- "user@nina.no" # insert NINA email
myPwd  <- "" # secret password
```

Choose which indicator(s) you want, use the NIcalc "importDatasetApi" function to retrieve data from the database and save the dataset locally.

```
indicator <- c("Dikesoldogg",
               "Jerv",
               "Elg",
               "Lomvi",
               "Havørn",
               "Lange")
```

```
for(i in indicator){
indicatorImport <- NULL
indicatorImport <- NIcalc::importDatasetApi(
  username = myUser,
  password = myPwd,
  indic = i,
  year = c("1990","2000","2010","2014","2019"))
```

```
assign(paste0(i, "_import"), indicatorImport)
}
```

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

for(i in indicator){
  temp <- get(paste0(i, "_import"))
  saveRDS(temp, paste0(path, i, "_import.rds"))
}

for(i in indicator){
  temp <- paste0(path, i, "_import.rds")
  assign(i, readRDS(temp))
}
```

Next, I need to assemble the data set. This shouldn't be necessary since all the data is already present. But One thing I notices was that for jerv, the distribution familiy and parameters only appear after assembling.

```
# Spesify all of Norway incl the five regions, som NIunits:
myNIunits <- c(allArea = T, parts = T, counties = F)
# Include all BSunits (kommuner) irrespective of the proportion of the main ecosystems
myPartOfTotal <- 0

for(i in indicator){

  temp <- get(paste0(i, "_import"))
  assemeble <- NULL
  assemeble <- NIcalc::assembleNiObject(
    inputData = temp,
    predefNIunits = myNIunits,
    partOfTotal = myPartOfTotal,
    indexType = "thematic",
    part = "ecosystem",
    total = "total")

  # I dont se the output changing if I for example chose total = marine. Perhaps 'part

  assign(paste0(i, "_assemble"), assemeble)

}
```

Save the files

```
for(i in indicator){
  temp <- get(paste0(i, "_assemble"))
```

```
  saveRDS(temp, paste0("data/", i, "_assemebled.rds"))
}
```

Loading the datafiles back into R.

```
for(i in indicator){
  temp <- paste0("data/", i, "_assemebled.rds")
  assign(i, readRDS(temp))
}
```

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


for(j in indicator){
print(j)

  temp <- get(j)
  temp2 <- get(paste0(j, "_import"))
  temp_comb <- data.frame(NULL)
  myMat2 <- NULL
  myMat2_comb <- NULL
  obstype <- NULL


  obstype <- temp$referenceValues$distributionFamilyName
  obstype[!is.na(obstype)] <- "tradObs"
  obstype[is.na(obstype)]  <- "customObs"

myMatr <- NIcalc::sampleObsMat(
  ICunitId           = temp$referenceValues$ICunitId,
  value              = temp$referenceValues$expectedValue,
  distrib            = temp$referenceValues$distributionFamilyName,
  mu                 = temp$referenceValues$distParameter1,
  sig                = temp$referenceValues$distParameter2,
  customDistribution = temp$referenceValues$customDistribution,
  obsType            = obstype,
  nsim =1000
        )

myMatr <- as.data.frame(myMatr)
myMatr <- myMatr %>%
  tibble::add_column(.before=1,
    ICunitID = row.names(myMatr))

myMatr <- myMatr %>%
  tibble::add_column(.after = 1,
```

```
      year = NA)

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

obs <- NULL
  obs <- temp$indicatorValues[[i]]$distributionFamilyName
  obs[!is.na(obs)] <- "tradObs"
  obs[is.na(obs)]  <- "customObs"


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

)


myMat2 <- as.data.frame(myMat)

myMat2 <- myMat2 %>%
  tibble::add_column(.before=1,
    ICunitID = row.names(myMat))

myMat2 <- myMat2 %>%
  tibble::add_column(.after = 1,
    year = myYears[i])

myMat2_comb <- rbind(myMat2_comb, myMat2)

 }

comb <- rbind(myMatr, myMat2_comb)

comb <- comb %>%
  tibble::add_column(.after = 1,
    ICunitName = temp2$ICunits$name[match(
      comb$ICunitID, temp2$ICunits$id)])
```

```
comb2 <- comb[!is.na(comb$year),]
comb3 <- comb[is.na(comb$year),]
comb3$ref_mean <- rowMeans(comb3[,-c(1:3)])

combScaled <- comb2 %>%
  tidyr::pivot_longer(cols = starts_with("V"))

combScaled <- combScaled %>%
  tibble::add_column(ref = comb3$ref_mean[
    match(combScaled$ICunitID, comb3$ICunitID)])
combScaled$scaledIndicator <- combScaled$value/combScaled$ref
combScaled <- dplyr::select(combScaled,
                      -name,
                      -value,
                      -ref)

assign(paste0(j, "_bootstrapped_raw"), comb)
assign(paste0(j, "_bootstrapped_scaled"), combScaled)

}
```

The reference values are also bootstrapped with uncertainties. These are coded as year = NA. We might, however, just end up using the row means.

Save the files

```
for(i in indicator){
  temp <- get(paste0(i, "_bootstrapped_raw"))
    temp <- get(paste0(i, "_bootstrapped_scaled"))

  saveRDS(temp, paste0("data/", i, "_bootstrapped_raw.rds"))
    saveRDS(temp, paste0("data/", i, "_bootstrapped_scaled.rds"))

}
```

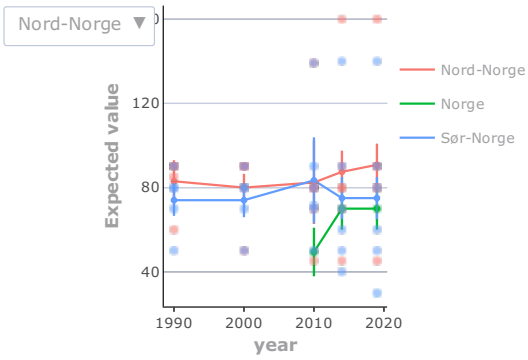
Chapter 3

Time series

3.1 Raw data

```
library(plotly)
## TODO change to correct data source
passerinesImport <- readRDS("data/passerinesImport.rds")

dat=passerinesImport$indicatorObservations$indicatorValues

dat$yearName=as.numeric(dat$yearName) # convert character vector to numeric years
sum_dat=dat %>%
  group_by(ICunitName, yearName) %>%
  summarise(mnExpected=mean(expectedValue, na.rm=TRUE),
            mnUpper=mean(upperQuantile, na.rm=TRUE),
            mnLower=mean(lowerQuantile, na.rm=TRUE)) # summerise the data to mean values

p=sum_dat %>%
  ggplot(aes(as.numeric(yearName), mnExpected, col=ICunitName))+
  geom_line()+
  geom_pointrange(aes(x=as.numeric(yearName), y=mnExpected, ymin=mnLower, ymax=mnUpper))+
  geom_point(data=dat, aes(as.numeric(yearName), expectedValue, alpha=0.2))+
  labs(x="year", y="Expected value")+
  theme_NIseries()
p2=ggplotly(p)
p2 %>% layout(
  updatemenus = list(
    list(
      type = "list",
      label = 'Category',
```

```
      buttons = list(
        list(method = "restyle",
             args = list('visible', c(TRUE, FALSE, FALSE)),
             label = "Nord-Norge"),
        list(method = "restyle",
             args = list('visible', c(FALSE, TRUE, FALSE)),
             label = "Norge"),
        list(method = "restyle",
             args = list('visible', c(FALSE, FALSE, TRUE)),
             label = "Sør-Norge")
      )
    )
  )
) # Add drop down menus for the data
```

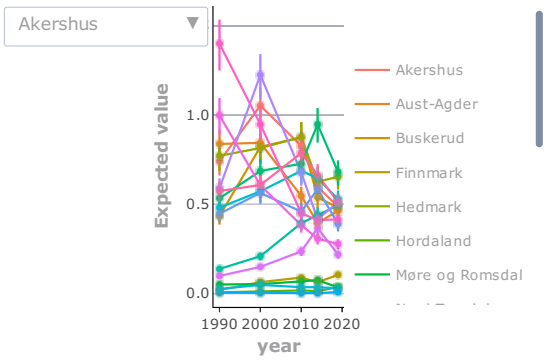


3.2 Scaled data

```
Elg_assemebled<- readRDS("data/Elg_assemebled.rds")
mycols=c("ICunitName" ,"yearName", "expectedValue","lowerQuantile", "upperQuantile")

data_list<-lapply(Elg_assemebled$indicatorValues, function(x) x%>% select(mycols))

dat<-bind_rows(data_list, .id = "column_label")
```

```
# plot
sum_dat=dat %>%
  group_by(ICunitName, yearName) %>%
  summarise(mnExpected=mean(expectedValue, na.rm=TRUE),
            mnUpper=mean(upperQuantile, na.rm=TRUE),
            mnLower=mean(lowerQuantile, na.rm=TRUE)) # summerise the data to mean values
source("R/ggplotTheme.R")
p=sum_dat %>%
  ggplot(aes(as.numeric(yearName), mnExpected, col=ICunitName))+
  geom_line()+
  geom_pointrange(aes(x=as.numeric(yearName), y=mnExpected, ymin=mnLower, ymax=mnUpper))+
  geom_point(data=dat, aes(as.numeric(yearName), expectedValue, alpha=0.2))+
  labs(x="year", y="Expected value")+
  theme_NIseries()
p2=ggplotly(p)
p2 %>% layout(
  updatemenus = list(
    list(
      type = "list",
      label = 'Category',
      buttons = list(
        list(method = "restyle",
             args = list('visible', c(TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS
             label = unique(dat$ICunitName)[2]),
        list(method = "restyle",
             args = list('visible', c(FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS
             label = unique(dat$ICunitName)[8]),
        list(method = "restyle",
             args = list('visible', c(FALSE, FALSE,TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE
             label = unique(dat$ICunitName)[5]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,TRUE, FALSE, FALSE, FALSE, FALSE, FALS
             label = unique(dat$ICunitName)[18]),
        list(method = "restyle",
             args = list('visible', c(FALSE, FALSE, FALSE,FALSE, TRUE, FALSE, FALSE, FALSE, FALSE
             label = unique(dat$ICunitName)[3]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, TRUE, FALSE, FALSE, FALS
             label = unique(dat$ICunitName)[11]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, TRUE, FALSE, FALS
             label = unique(dat$ICunitName)[13]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE, TRUE, FALS
             label = unique(dat$ICunitName)[15]),
```

```
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[16]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[4]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[10]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[12]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[14]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[7]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[17]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[9]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[6]),
        list(method = "restyle",
             args = list('visible', c( FALSE, FALSE, FALSE,FALSE, FALSE, FALSE, FALSE,
             label = unique(dat$ICunitName)[1])
      )
    )
  )
) # Add drop down menus for the data
```


Akershus
Expected value
1.0
0.5
0.0
1990 2000 2010 2020
year
Akershus
Aust-Agder
Buskerud
Finnmark
Hedmark
Hordaland
Møre og Romsdal

Chapter 4

Maps

4.1 Raw data

4.2 Scaled data

4.2.1 Jerv

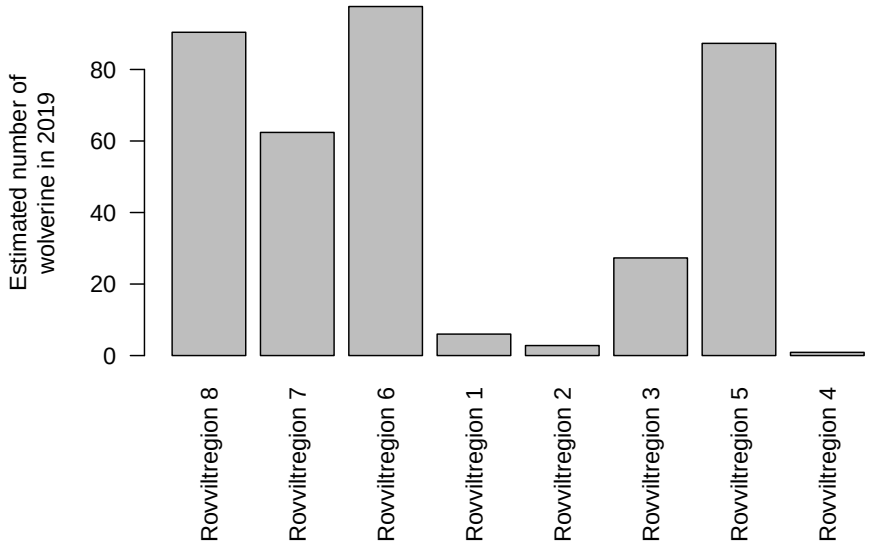
4.2.1.1 Prepare NI data

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_assemebled.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.

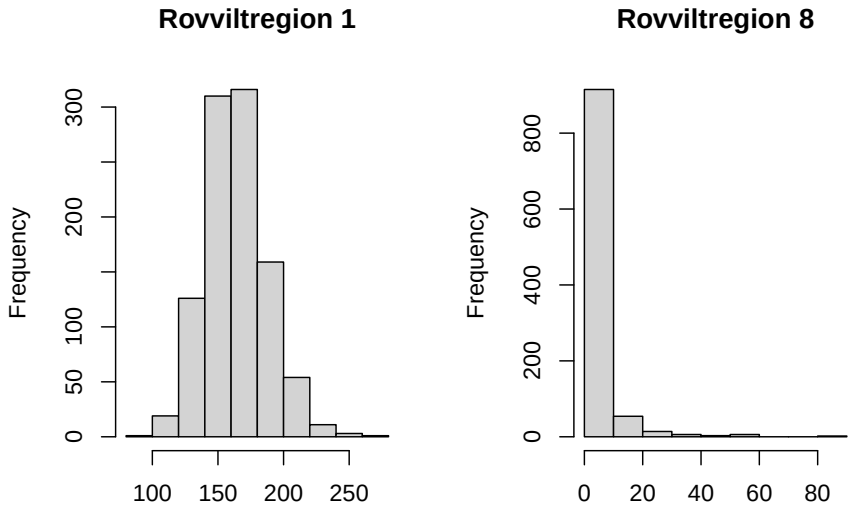
```
par(mar=c(9,5,1,1))
barplot(jerv$indicatorValues$`2019`$expectedValue,
        names.arg = jerv$indicatorValues$`2019`$ICunitName,
        las=2,
        ylab = "Estimated number of\nwolverine in 2019")
```



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 = "")
```

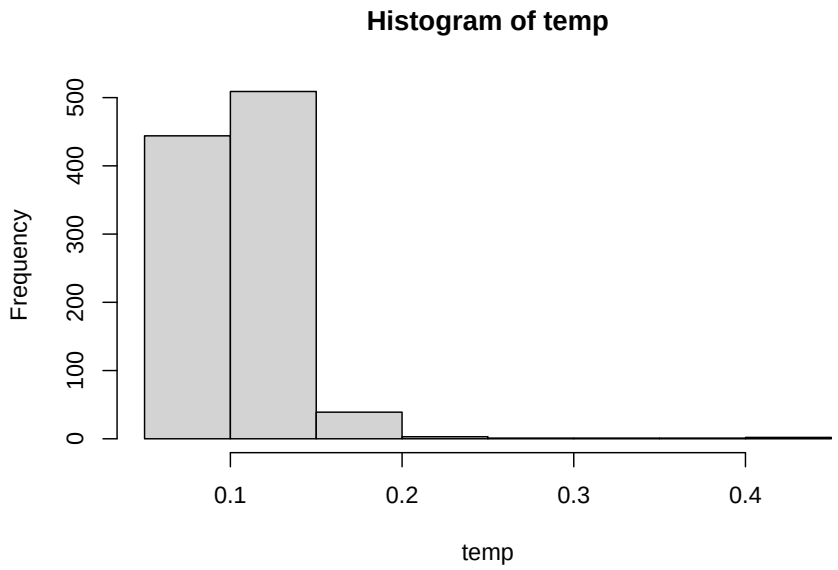


For some reason the extected values are far from the mean of these distributions. I did this exercise once before, and did not get this problem then. I think the difference is that I use eco = NULL this time, in the `importDatasetApi()`, and this cause the output to somehow split into forest and alpine 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
#>
#> Attaching package: 'matrixStats'
#> The following object is masked from 'package:dplyr':
#>
#>     count
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	163.76	23.55	848.52	0.19	0.14	Rovvaltregion 8
1304	59.57	22.09	349.04	0.17	0.37	Rovvaltregion 7
1305	12.46	20.16	405.34	0.03	1.62	Rovvaltregion 6
1311	3.52	9.71	282.74	0.01	2.76	Rovvaltregion 1
1313	3.68	10.5	203.53	0.02	2.85	Rovvaltregion 2
1315	10.23	52.24	193.93	0.05	5.11	Rovvaltregion 3
1316	4.12	11.23	165.9	0.02	2.73	Rovvaltregion 5
5141	3.41	7.91	28.68	0.12	2.32	Rovvaltregion 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)
```

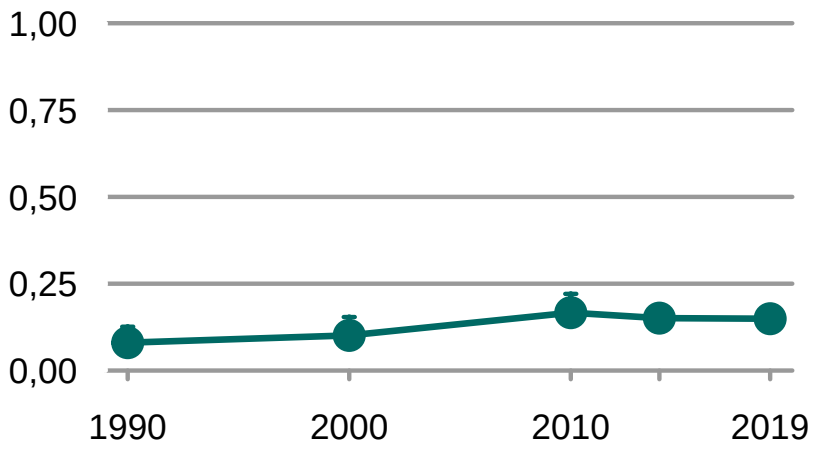


Figure 4.1: The scaled indicator values for wolverine across Norway.

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. The geometries for the appropriate spatial units of each indicator can be downloaded in .json format via a previously created API for the nature index database: https://ninweb08.nina.no/NaturindeksAPI/index.html To get the file for a specific indicator, one needs to enter the numerical indicator id under "/api/Indicator/{id}/Areas" and then click download. We then converted the .json file to shapefiles for use in R.

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

```
library(sf)
#> Warning: package 'sf' was built under R version 4.1.3
#> Linking to GEOS 3.10.2, GDAL 3.4.1, 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",]
```

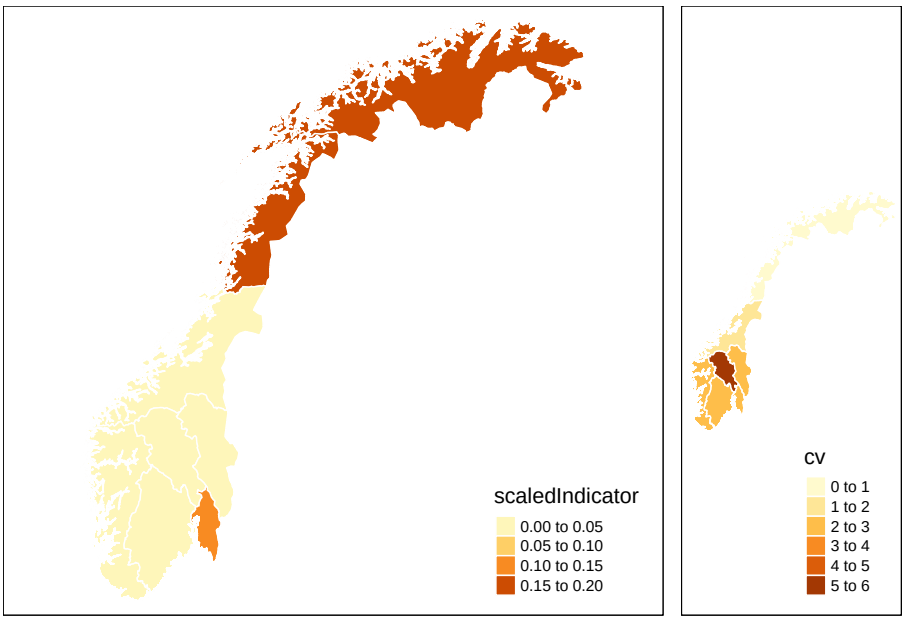
Clip it against the outline of Norway to make it look more pretty

```
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
```

```
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)
#> Warning: package 'tmap' was built under R version 4.1.3
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))
```


scaledIndicator
0.00 to 0.05
0.05 to 0.10
0.10 to 0.15
0.15 to 0.20
cv
0 to 1
1 to 2
2 to 3
3 to 4
4 to 5
5 to 6

Chapter 5

Other figures

5.1 Gradient density plots for interactive maps

The maps presented on the webpage do not include any representation of uncertainty. One way of including that information without having to add additional (layers to the) maps would be to build on the interactive functions included so far and present a probability distribution for the given region and year. This could be displayed in the same box that currently appears when hovering over an area and displays area name and average indicator value.

To make these density plots, we use the previously simulated bootstrap samples, using Jerv as an example:

```
i <- "Jerv"
bootStrp <- readRDS(paste0("data/", i, "_bootstrapped_scaled.rds"))
head(bootStrp)
#> # A tibble: 6 x 4
#>   ICunitID ICunitName      year  scaledIndicator
#>   <chr>    <chr>           <chr>           <dbl>
#> 1 1302     Rovviltregion 8 1990            0.194
#> 2 1302     Rovviltregion 8 1990            0.233
#> 3 1302     Rovviltregion 8 1990            0.190
#> 4 1302     Rovviltregion 8 1990            0.207
#> 5 1302     Rovviltregion 8 1990            0.199
#> 6 1302     Rovviltregion 8 1990            0.164
```

I have considered forcing all indicator values > 1 to display as 1, but this messes up when plotting density functions. Still, this conversion is required for calculating the point estimate (median) and I therefore make a copy of the data in which no values are larger than 1.

```
bootStrp1 <- bootStrp
bootStrp1$scaledIndicator[which(bootStrp1$scaledIndicator > 1)] <- 1
```

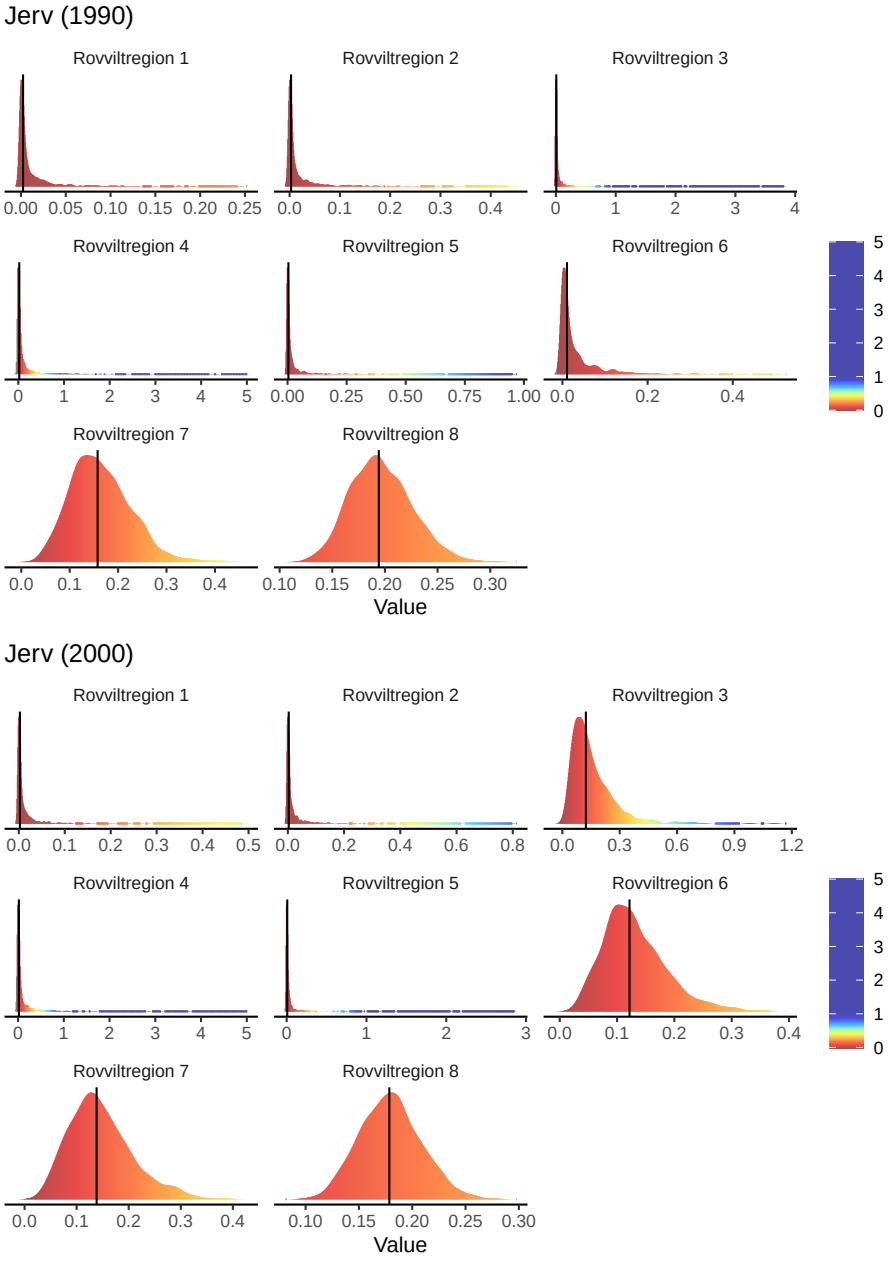
Next, we need to manually calculate the probability densities for the indicator values in each year and area. This is necessary for making density plots with a color gradient fill (but see further below for an alternative using the "ggridges" package which does not require this intermediate step).

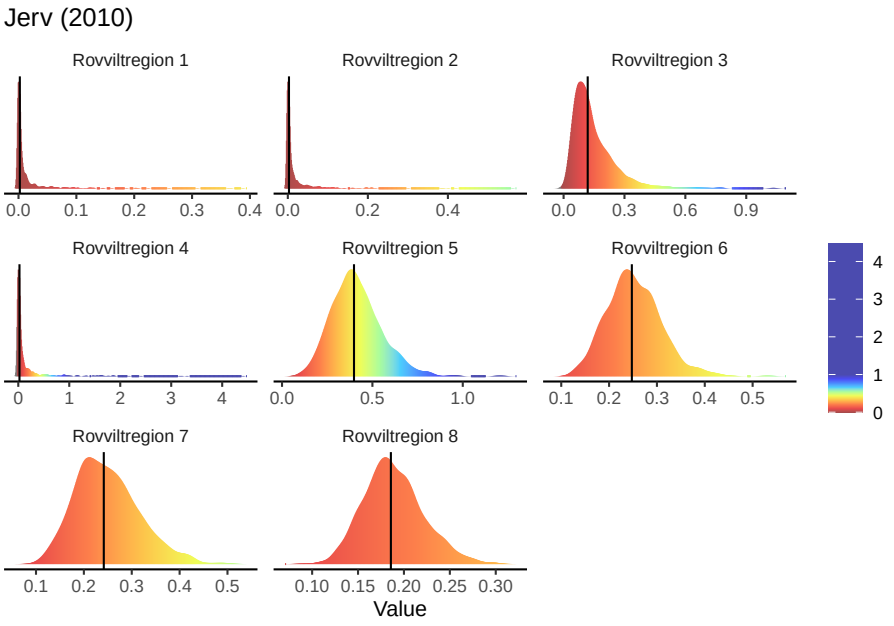
```
years <- unique(bootStrp$year)
areas <- unique(bootStrp$ICunitName)

pDens <- data.frame()
for(t in years){
  for(a in areas){

    bootStrp_sub <- subset(bootStrp, year == t & ICunitName == a)

    pDens_a_t <- data.frame(
      ICunitName = a,
      year = t,
      x = density(bootStrp_sub$scaledIndicator)$x,
      y = density(bootStrp_sub$scaledIndicator)$y
    )

    pDens <- rbind(pDens, pDens_a_t)
  }
}
```

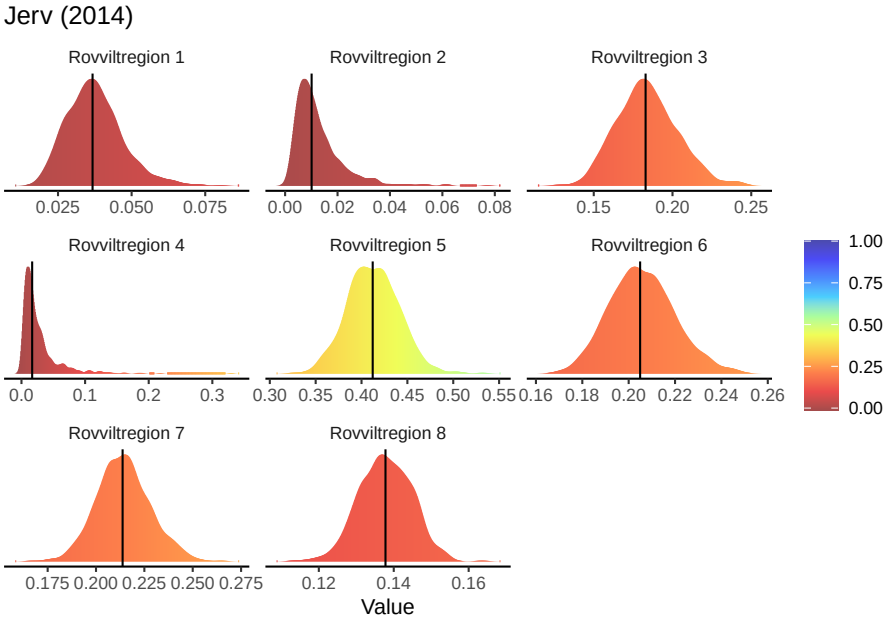
We can then proceed to plotting the probability density functions with a color gradient under the line:

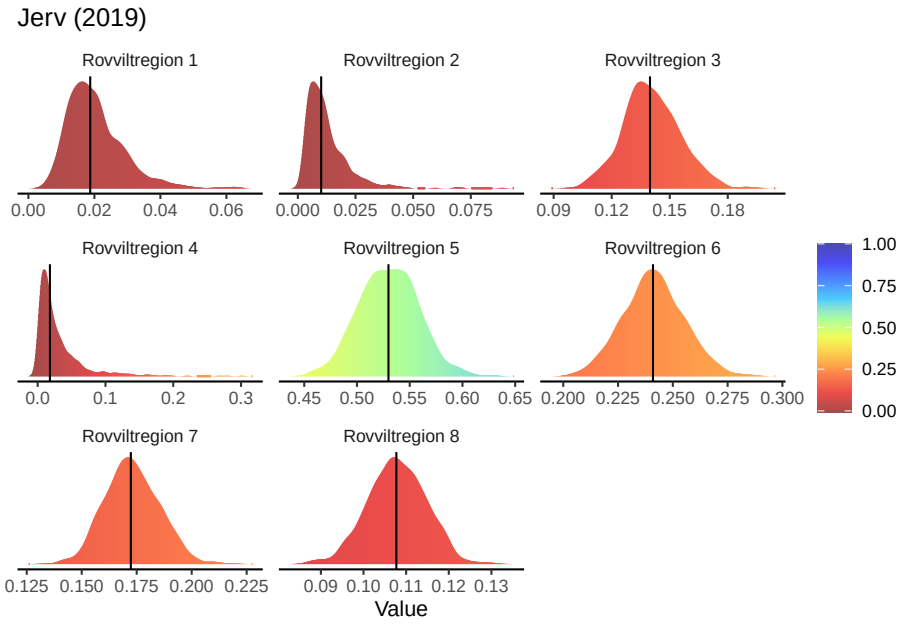
```
# Set maximum value (we will not plot beyond 4)

for(t in years){

  # Take data subsets for a given year
  bootStrp_yr <- bootStrp[which(bootStrp$year == t),]
  bootStrp1_yr <- bootStrp1[which(bootStrp1$year == t),]
  pDens_yr <- pDens[which(pDens$year == t),]

  # Extract distribution medians
  sum_values <- bootStrp1_yr %>%
    group_by(ICunitName) %>%
    summarise(sumStat = median(scaledIndicator))

  # Set maximum plotting value (never > 5) and mapping for custom color scale
```

```
  maxVal <- ifelse(max(pDens_yr$x) > 5, 5, max(pDens_yr$x))

  if(maxVal < 1+1/9){
    valuesMap <- c(-0.1, seq(0, 1, length.out = 10))
    colorMap <- c("#1F8C81", NIviz_colours$IndMap_cols)
  }else{
    valuesMap <- c(-0.1, c(seq(0, 1, length.out = 10), maxVal)/maxVal)
    colorMap <- c("#1F8C81", NIviz_colours$IndMap_cols, "#4B4BAF")
  }

  # Plot densities
  print(
    ggplot(subset(pDens_yr, x <= maxVal), aes(x, y)) +
      geom_segment(aes(xend = x, yend = 0, colour = x)) +
      #scale_color_NIviz_c(name = "IndMap_cols") +
      scale_colour_gradientn(colours = colorMap,
                             values = valuesMap,
                             limits = c(-0.01, ifelse(maxVal < 1, 1, maxVal))) +
      ggtitle(paste0(i, " (", t, ")")) +
      xlab("Value") +
      geom_vline(data = sum_values, aes(xintercept = sumStat)) +
      facet_wrap(~ ICunitName, scales = 'free') +
      theme_classic() +
      theme(strip.background = element_blank(),
            legend.title = element_blank(),
            axis.line.y = element_blank(), axis.ticks.y = element_blank(),
            axis.text.y = element_blank(), axis.title.y = element_blank())
  )
}
```


Jerv (1990)
Rovviltregion 1
Rovviltregion 2
Rovviltregion 3
Rovviltregion 4
Rovviltregion 5
Rovviltregion 6
Rovviltregion 7
Rovviltregion 8
Value
Jerv (2000)
Rovviltregion 1
Rovviltregion 2
Rovviltregion 3
Rovviltregion 4
Rovviltregion 5
Rovviltregion 6
Rovviltregion 7
Rovviltregion 8
Value


Jerv (2010)
Rovviltregion 1
Rovviltregion 2
Rovviltregion 3
Rovviltregion 4
Rovviltregion 5
Rovviltregion 6
Rovviltregion 7
Rovviltregion 8
Value

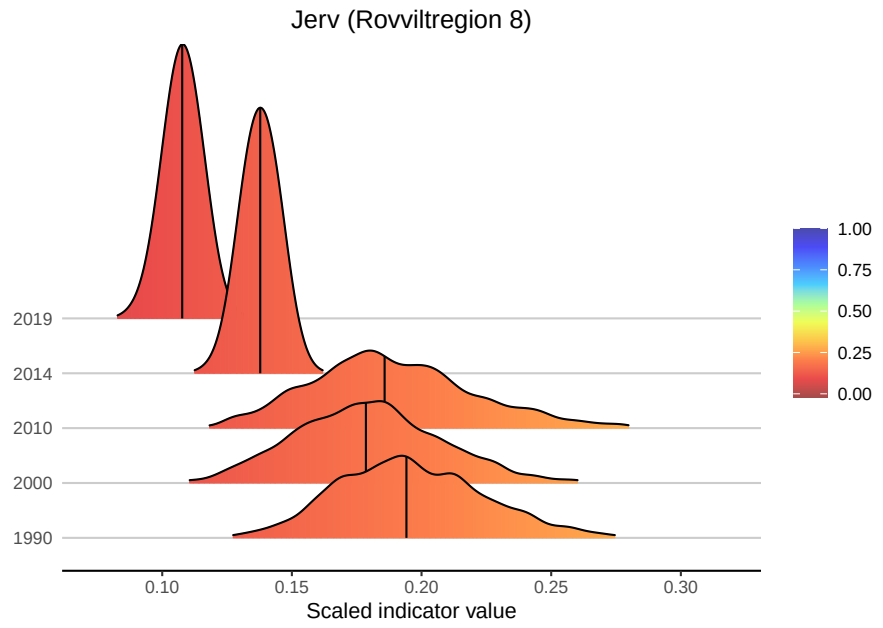

Jerv (2014)
Rovviltregion 1
Rovviltregion 2
Rovviltregion 3
Rovviltregion 4
Rovviltregion 5
Rovviltregion 6
Rovviltregion 7
Rovviltregion 8
Value



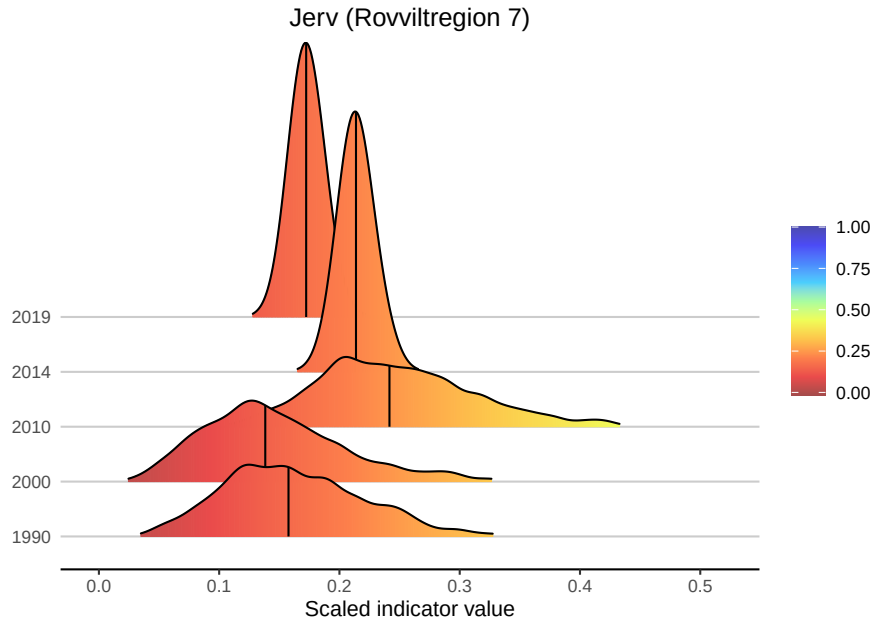
Gradient density plots might also be useful for other types of visualization of indicator and index data. For example, there is a very attractive way of using ridgeplots for visualizing time series inclusing uncertainty. Could look something like this:

```
for(a in areas){

  # Take data subsets for a given area
  bootStrp_ar <- bootStrp[which(bootStrp$ICunitName == a),]
  pDens_ar <- pDens[which(pDens$ICunitName == a),]

  # Set maximum plotting value (never > 5) and mapping for custom color scale
  maxVal <- ifelse(max(pDens_ar$x) > 5, 5, max(pDens_ar$x))

  if(maxVal < 1+1/9){
    valuesMap <- c(-0.1, seq(0, 1, length.out = 10))
    colorMap <- c("#1F8C81", NIviz_colours$IndMap_cols)
  }else{
    valuesMap <- c(-0.1, c(seq(0, 1, length.out = 10), maxVal)/maxVal)
    colorMap <- c("#1F8C81", NIviz_colours$IndMap_cols, "#4B4BAF")
  }

  # Plot densities
  print(
```

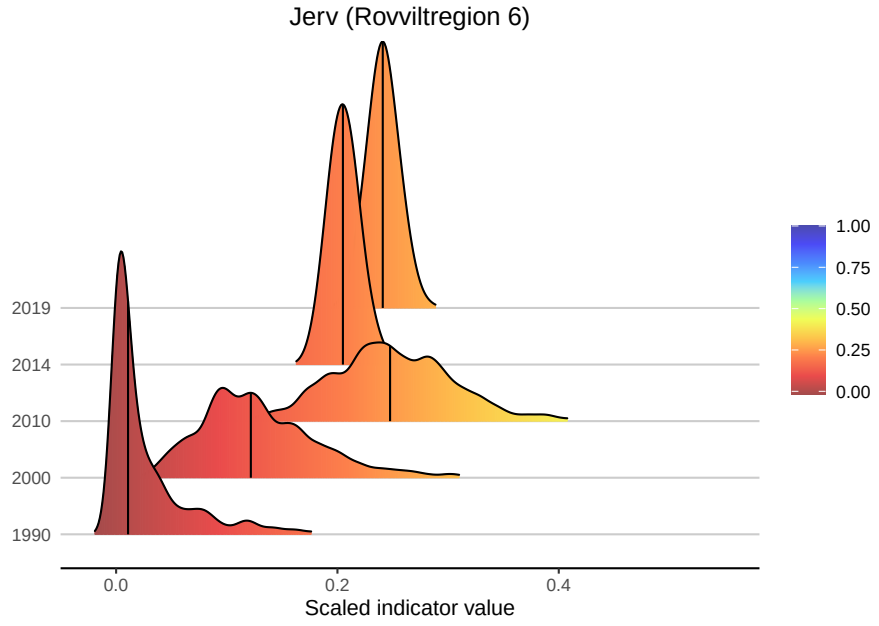
```
  ggplot(subset(bootStrp_ar, scaledIndicator <= maxVal), aes(x = scaledIndicator, y = year, fill
    geom_density_ridges_gradient(scale = 5, rel_min_height = 0.01, quantile_lines = TRUE, quantil
    scale_fill_gradientn(colours = colorMap,
                         values = valuesMap,
                         limits = c(-0.02, ifelse(maxVal < 1, 1, maxVal))) +
    ggtitle(paste0(i, " (", a, ")")) +
    xlab("Scaled indicator value") + ylab("") +
    theme_classic() +
    theme(legend.title = element_blank(), plot.title = element_text(hjust = 0.5),
          axis.line.y = element_blank(), axis.ticks.y = element_blank(),
          panel.grid.major.y = element_line(color = "grey80"))
  )

}
#> Picking joint bandwidth of 0.00457
```



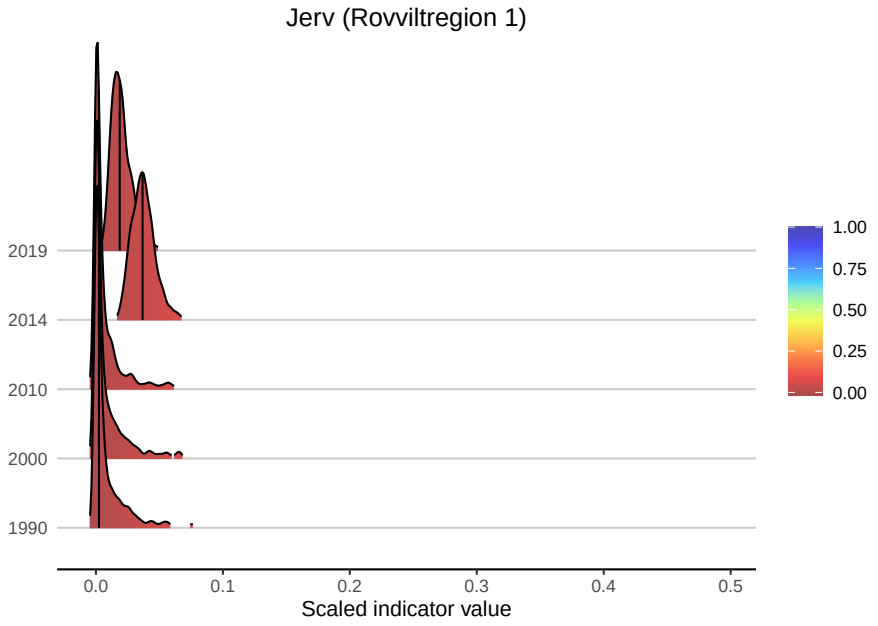

```
#> Picking joint bandwidth of 0.00947
```



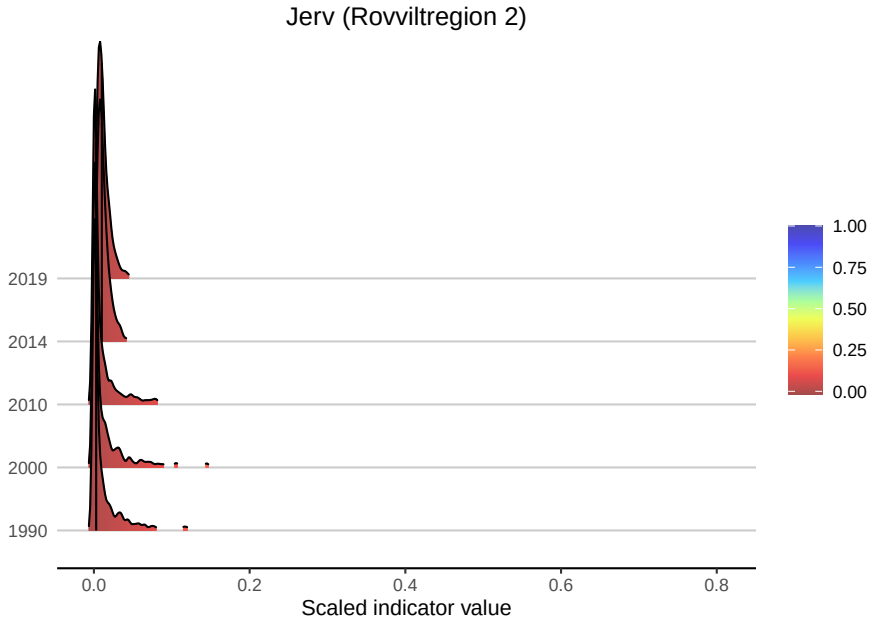

```
#> Picking joint bandwidth of 0.00715
```



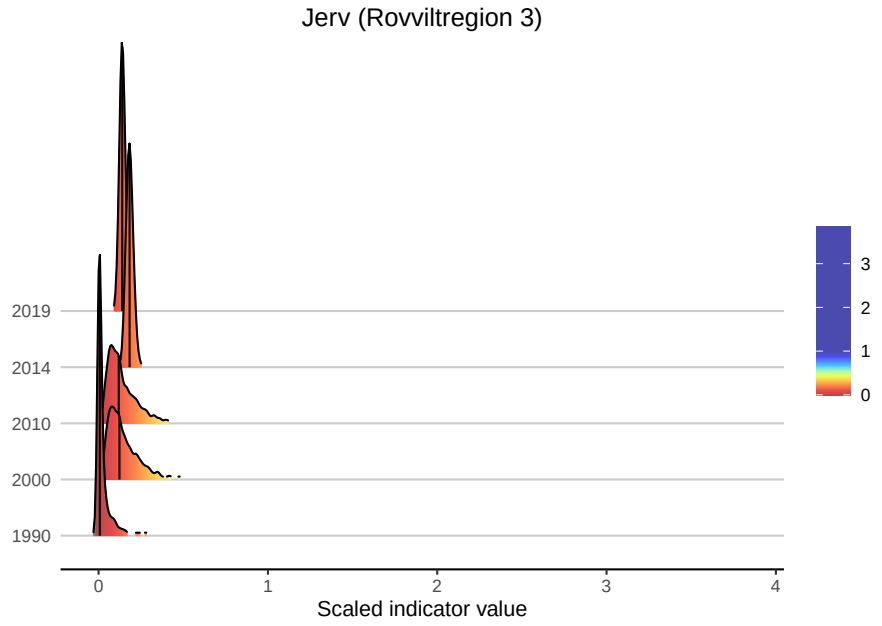

```
#> Picking joint bandwidth of 0.00184
```



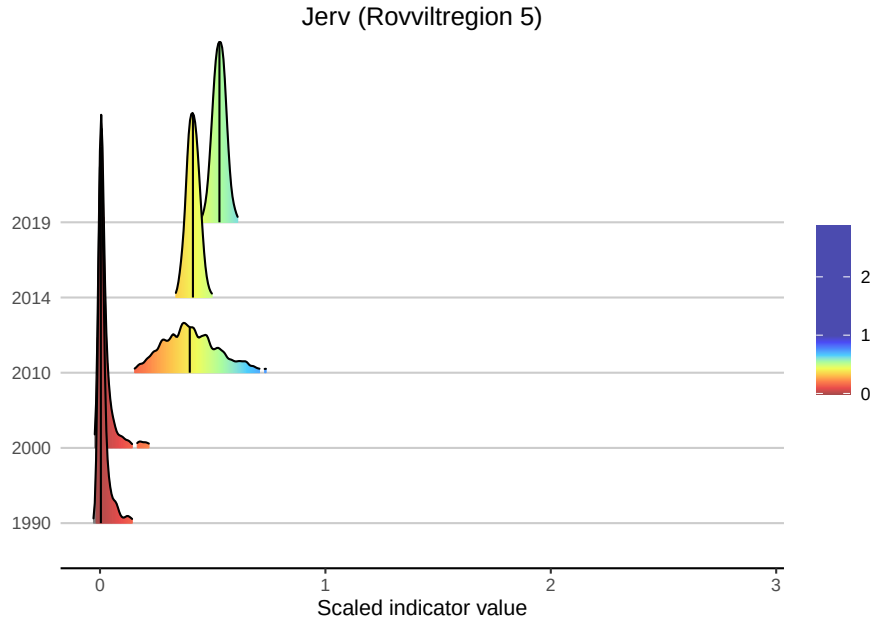

```
#> Picking joint bandwidth of 0.00215
```



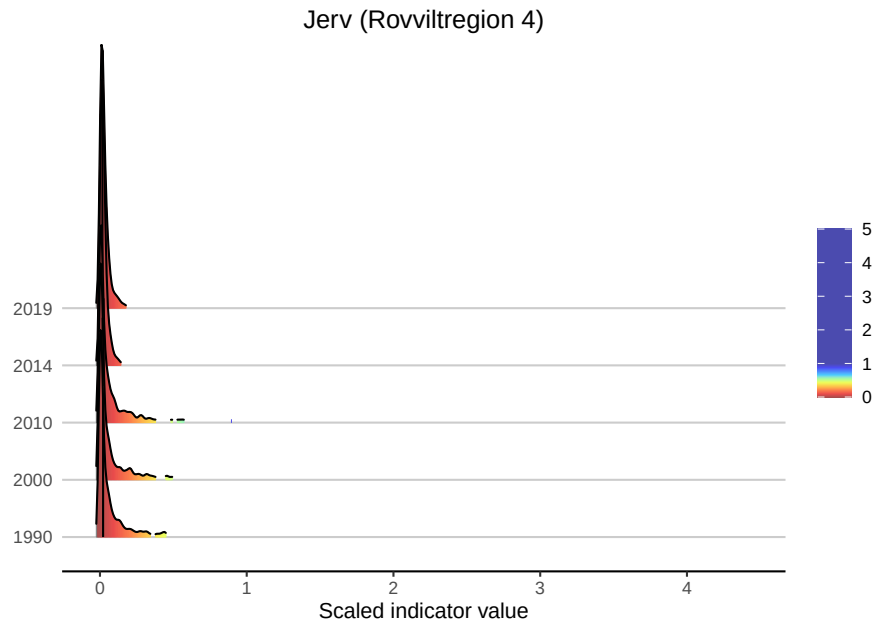

```
#> Picking joint bandwidth of 0.0101
```




```
#> Picking joint bandwidth of 0.00966
```




```
#> Picking joint bandwidth of 0.0112
```



5.2 Ecosystem fidelity

All indicators are assigned to at least one ecosystem, but a fair number of them are assigned to multiple ecosystems by means of proportions. Wolverine (Jerv), for example, is assigned with 25% to forest and and 75% to mountain. This basic information could easily be displayed on each indicator's page on naturindeks.no.

The relevant information is found in the assembled indicator data under $indicators:

```
i <- "jerv"
indexData <- readRDS(paste0("data/", i, "_assemebled.rds"))

str(indexData$indicators)
#> 'data.frame':    1 obs. of  9 variables:
#>  $ id                : num 88
#>  $ name              : chr "Jerv"
#>  $ keyElement        : logi FALSE
#>  $ functionalGroup   : chr "Topp-predator generalist"
#>  $ functionalGroupId: num 8
#>  $ scalingModel      : chr "Low"
#>  $ scalingModelId    : num 1
#>  $ Fjell             : num 75
```

```
#>  $ Skog              : num 25
```

Any ecosystem type relevant to a specific indicator appears as a separate column in this dataframe, and contains a value representing the % fidelity to that ecosystem type.

Using separately stored information on available ecosystem types, we can assemble this data for all of our example indicators:

```
# Load ecosystem info
EcoSysInfo <- readRDS("data/EcosystemInfo.rds")

# Indicator list
indicator <- c("Dikesoldogg",
               "Jerv",
               "Elg",
               "Lomvi",
               "Havørn",
               "Lange")

# Assemble fidelity data
fidData <- data.frame()

for(i in 1:length(indicator)){

  indexData <- readRDS(paste0("data/", indicator[i], "_assemebled.rds"))

  ColIdx <- which(names(indexData$indicators) %in% EcoSysInfo$ecosystem)

  fidDataI <- data.frame(
    indicator = indicator[i],
    ecosystem = names(indexData$indicators)[ColIdx],
    fidelity = unname(as.numeric(indexData$indicators[,ColIdx])))

  fidData <- rbind(fidData, fidDataI)
}
```

This gives us a dataframe with all indicators and their fidelity to different ecosystems:

```
print(fidData)
#>     indicator         ecosystem fidelity
#> 1 Dikesoldogg           Våtmark      100
#> 2        Jerv             Fjell       75
#> 3        Jerv              Skog       25
#> 4         Elg              Skog      100
#> 5       Lomvi      Hav-pelagisk       67
```

```
#> 6       Lomvi Kystvann-pelagisk       33
#> 7      Havørn Kystvann-pelagisk      100
#> 8       Lange           Havbunn       80
#> 9       Lange     Kystvann-bunn       20
```

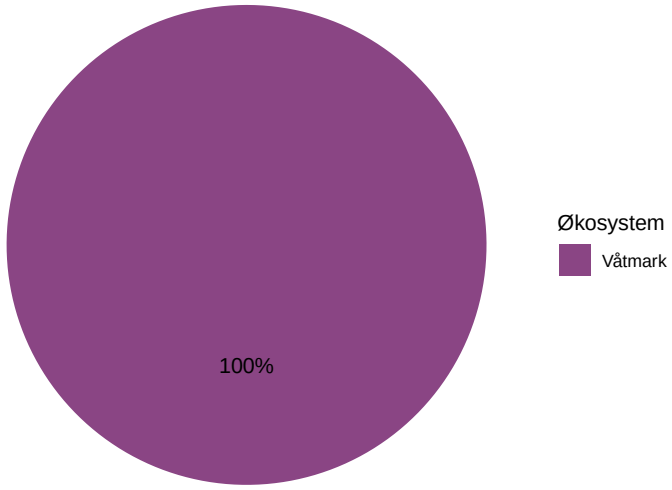
Before plotting, we match the integer ecosystem IDs to make sure the colour mapping works correctly:

```
fidData <- merge(fidData, EcoSysInfo, all.x = TRUE)
```

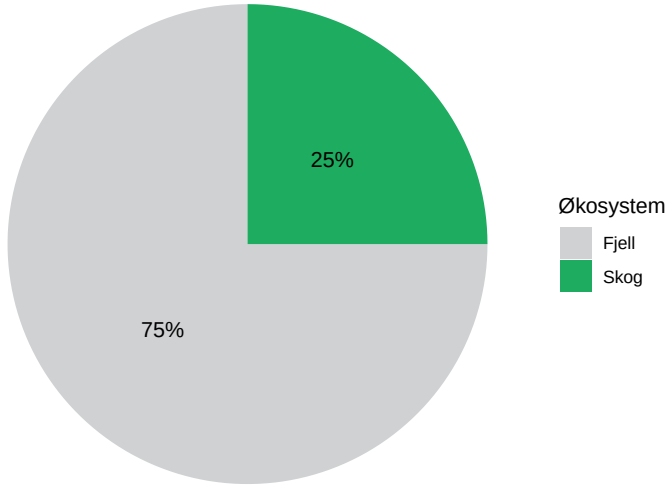
Next, we'll visualize this information for each indicator by means of pie charts.

```
EcoSys_cols <- NIviz_colours$EcoSys_cols[1:11]
names(EcoSys_cols) <- EcoSysInfo$ecosystem

for(i in 1:length(indicator)){

  sub_fidData <- fidData[which(fidData$indicator == indicator[i]),]

  print(
    ggplot(sub_fidData, aes(x = "", y = fidelity, fill = fct_inorder(ecosystem))) +
    ggtitle(indicator[i]) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(fidelity, "%")),
              position = position_stack(vjust = 0.5)) +
    coord_polar(theta = "y") +
    scale_fill_manual(name = "Økosystem", values = EcoSys_cols[which(names(EcoSys_cols) %in% sub_
    theme_void()
  )

}
```

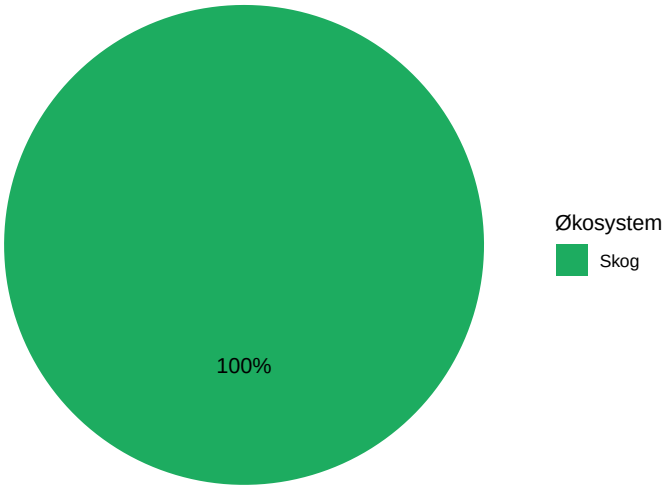
Dikesoldogg



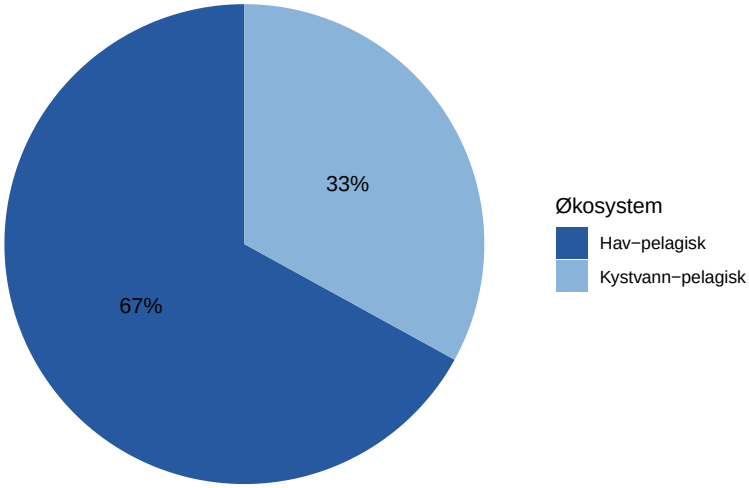
Jerv

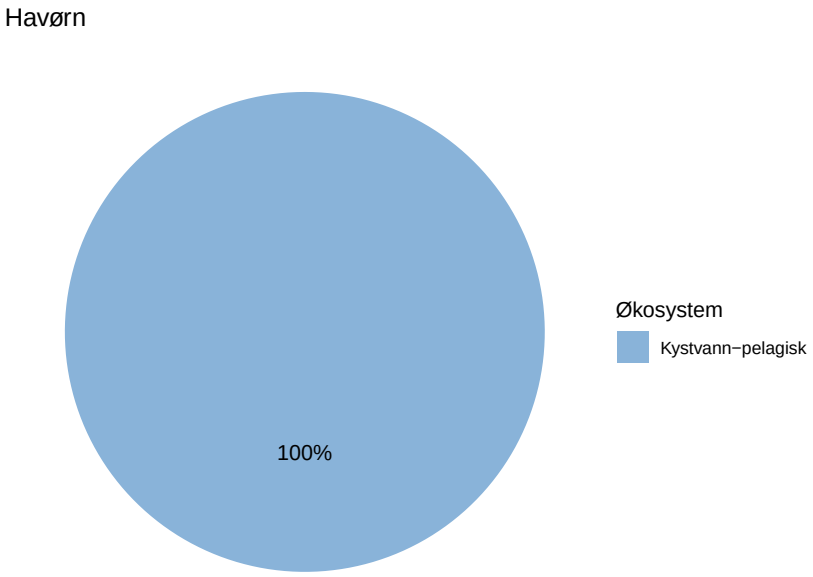


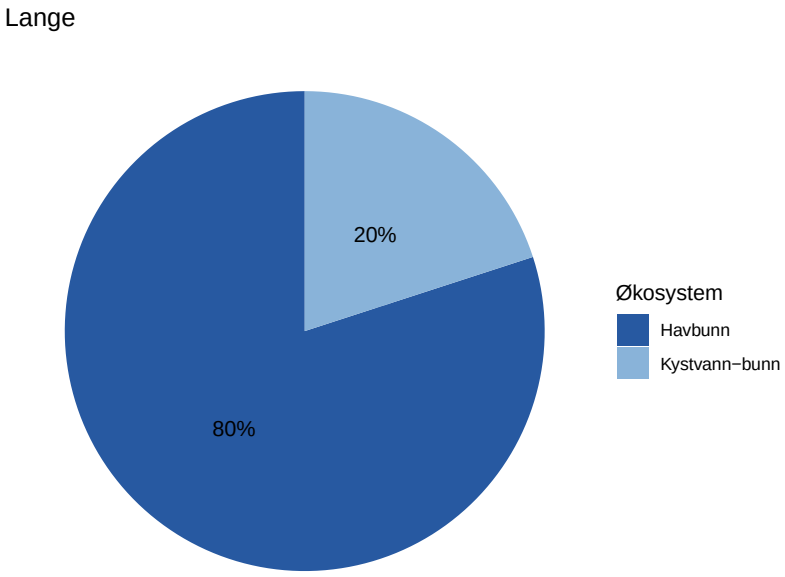
Elg


100%
Økosystem
Skog

Lomvi


33%
67%
Økosystem
Hav−pelagisk
Kystvann−pelagisk



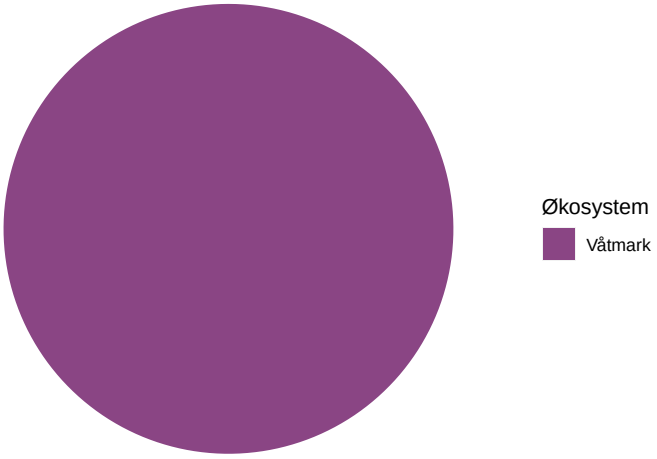


Depending on how large these pie charts should appear on the website in the end, it may be necessary to move the percentage labels outside the pies. Doing that is a bit more cumbersome, but works with the code below unless the indicator belongs 100 % to one dataset. When that is the case, the solution below does not print the percentage label at all (see indicator Lange) and I

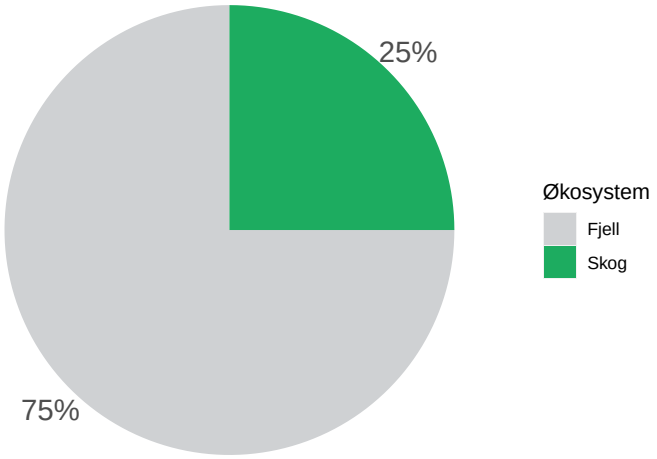
have not figured out how to fix that yet.

```
for(i in 1:length(indicator)){

  sub_fidData <- fidData[which(fidData$indicator == indicator[i]),]

  posData <- sub_fidData %>%
  mutate(csum = rev(cumsum(rev(fidelity))),
         pos = fidelity/2 + lead(csum, 1),
         pos = if_else(is.na(pos), fidelity/2, pos))

  print(
  ggplot(sub_fidData, aes(x = "", y = fidelity, fill = ecosystem)) +
    ggtitle(indicator[i]) +
    geom_bar(stat = "identity") +
    coord_polar(theta = "y") +
    #scale_fill_manual(values = EcoSys_cols) +
    scale_fill_manual(name = "Økosystem", values = EcoSys_cols[which(names(EcoSys_cols) %in% sub_
    scale_y_continuous(breaks = posData$pos, labels = paste0(sub_fidData$fidelity, "%")) +
    theme(axis.ticks = element_blank(),
          axis.title = element_blank(),
          axis.text = element_text(size = 15),
          panel.background = element_rect(fill = "white"))
  )

}
```

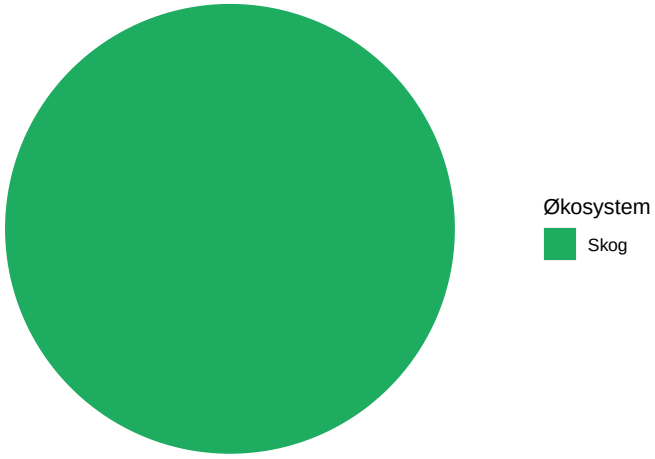
Dikesoldogg



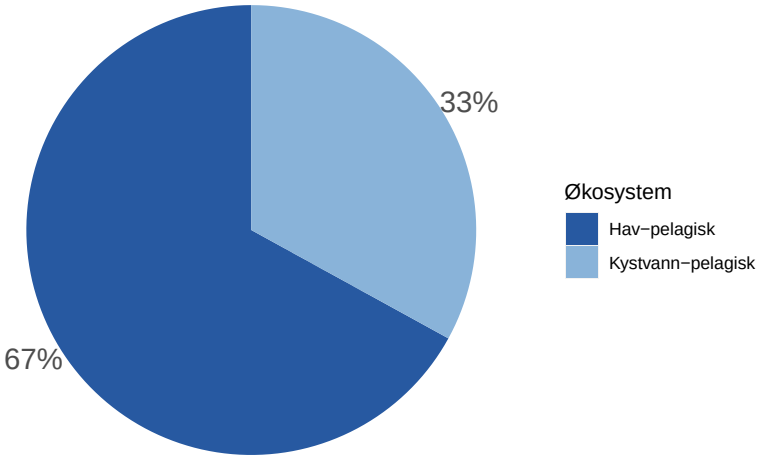
Jerv

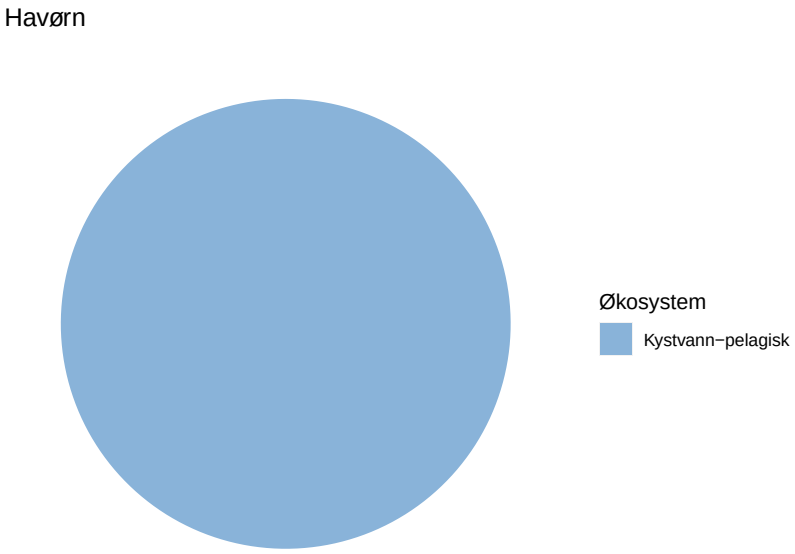


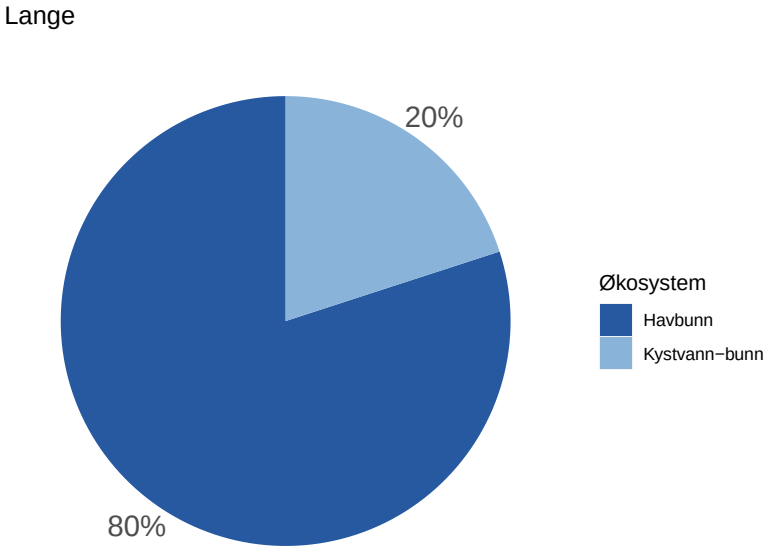
Elg


Økosystem
Skog

Lomvi


33%
67%
Økosystem
Hav−pelagisk
Kystvann−pelagisk





5.3 Impact factor wordclouds

```
library(wordcloud)
```

```
library(NIcalc)
library(wordcloud2)
library(RColorBrewer)
library(readxl)
library(dplyr)
library(tm)
```

As per now, the website shows the most important impact factors for each indicator as a list in a pop-up window on the top-right. This is information that is likely of great interest to the general public, and would benefit from being placed more visible on the website and in a more engaging way than as a list. One attractive alternative way of presenting is via wordclouds.

The following wordcloud figures show the pressure factors for each indicator, where both the color saturation and text size represent the importance of each pressure factor. Small text size and low saturation means low importance, while large text size represent pressure factors with high importance to the indicator.

```
# Load dataset
pressure = read_excel("P:/41201612_naturindeks_2021_2023_database_og_innsynslosning/Pilot_Forbedr

# Filter for species of interest
pressure = pressure %>% filter(navn_norsk == "Elg" | navn_norsk == "Dikesoldogg" | navn_norsk ==
pressure = arrange(pressure, by = navn_norsk)
pressure = pressure %>% rename(PressureFactor = Paavirkningsfaktor, PressureValue = FK_Paavirknin

# Remove all instances of "Ikke rel/ukjent" category" and increase the value of pressure factors
pressure = pressure %>% filter(PressureValue != 7) %>% mutate(PressureValue = PressureValue*2)

# Create separate datasets for the different species
dikesoldogg = pressure %>% filter(navn_norsk == "Dikesoldogg") %>% dplyr::select(PressureFactor,

elg = pressure %>% filter(navn_norsk == "Elg") %>% dplyr::select(PressureFactor, PressureValue)

havørn = pressure %>% filter(navn_norsk == "Havørn") %>% dplyr::select(PressureFactor, PressureVa

lange = pressure %>% filter(navn_norsk == "Lange") %>% dplyr::select(PressureFactor, PressureValu

lomvi = pressure %>% filter(navn_norsk == "Lomvi") %>% dplyr::select(PressureFactor, PressureValu

# Create a custom color gradient (green in this case)
pressureColors = c("#bde4aa", "#addb9d", "#9cd28f", "#8cc982", "#7dc275", "#6cb967", "#5db15a", "
```

```
# Dikesoldogg
wordcloud(words = dikesoldogg$PressureFactor, freq = dikesoldogg$PressureValue, min.fre
```

5.3.1 Dikesoldogg (oblong-leaved sundew)


```
wordcloud(words = elg$PressureFactor, freq = elg$PressureValue, min.freq = 1, max.words
```

5.3.2 Elg (Moose)

5.3.3 Havørn (White-tailed eagle)

```
wordcloud(words = havørn$PressureFactor, freq = havørn$PressureValue, min.freq = 1, max.words = 2
```

```
wordcloud(words = lange$PressureFactor, freq = lange$PressureValue, min.freq = 1, max.w
```

5.3.4 Lange (Common ling)


```
wordcloud(words = lomvi$PressureFactor, freq = lomvi$PressureValue, min.freq = 1, max.w
```

5.3.5 Lomvi (common guillemot)

5.4 Data type

```
source("R/colorPalettes.R")
```

```
#Elg datatyper
data <- data.frame(
  category=c("Ekspert",
             "Modeller",
             "Overvåkning"),
  count=c(4.9, 95.1, 0)
)

# load library
library(tidyverse)
# Compute percentages
data$fraction <- data$count / sum(data$count)

# Compute the cumulative percentages (top of each rectangle)
data$ymax <- cumsum(data$fraction)

# Compute the bottom of each rectangle
data$ymin <- c(0, head(data$ymax, n=-1))

# Compute label position
```

```
data$labelPosition <- (data$ymax + data$ymin) / 2
data$labelPosition[3]<-0.8
# Compute a good label
data$label <- paste0(data$category, "\n ", data$count, " %")
library(ggrepel)
#> Warning: package 'ggrepel' was built under R version 4.1.3
# Make the plot
ggplot(data, aes(ymax=ymax, ymin=ymin, xmax=4, xmin=3, fill=category)) +
  geom_rect() +
  geom_text(x=2, aes(y=labelPosition, label=label, color=category), size=2.5) + # x he
  scale_fill_NIviz_d("IndMap_cols") +
  scale_colour_NIviz_d("IndMap_cols") +
  coord_polar(theta="y") +
  xlim(c(-1, 4)) +
  theme_void() +
  theme(legend.position = "none")
```