NIvis

2022-10-14

Contents

Chapter 1

Introduction

Chapter 2

Downloading and preparing example data

```
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"))

}
```

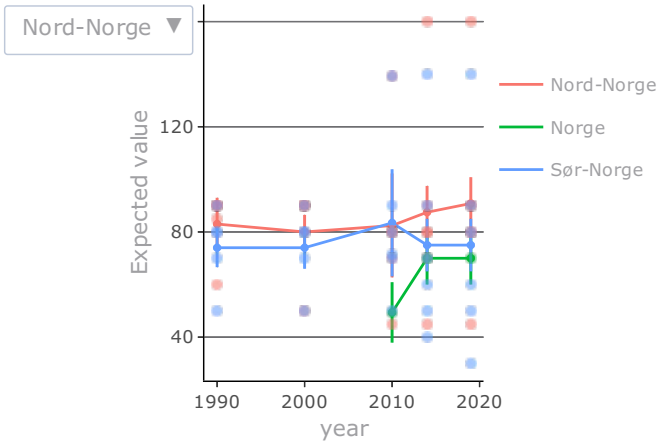
Chapter 3

Time series

3.1 Raw data

```
library(plotly)
## TODO change to correct data source
passerinesImport <- readRDS(paste0(here::here(), "/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_classic()+
  theme(text = element_text(colour = "#A0A0A3"),
        title = element_text(colour = "#FFFFFF"),
axis.title.x = element_text(colour = "#A0A0A3"),
axis.title.y = element_text(colour = "#A0A0A3"),
panel.grid.major.y = element_line(colour = "#707073", size=0.2)) # replace this with the correct
```

```
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

Chapter 4

Maps

4.1 Raw data

4.2 Scaled data

4.2.1 Jerv

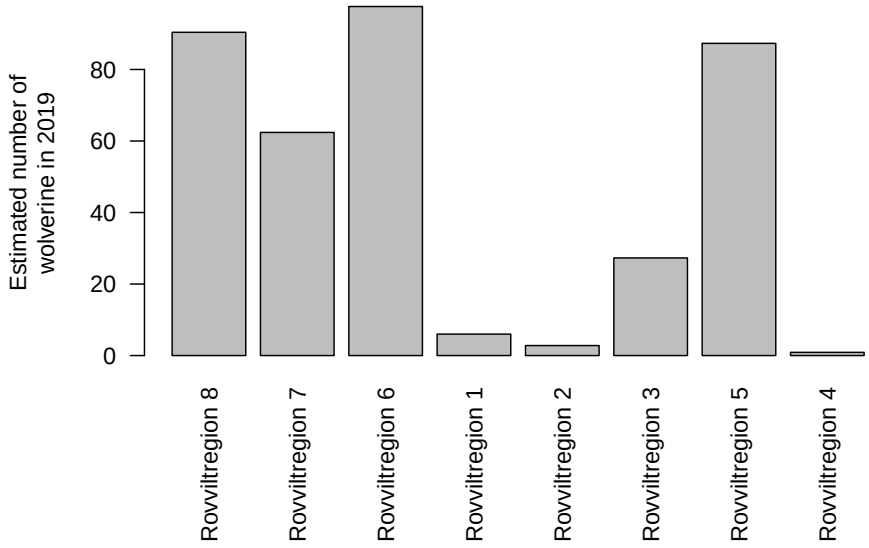
4.2.1.1 Prepare NI data

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.

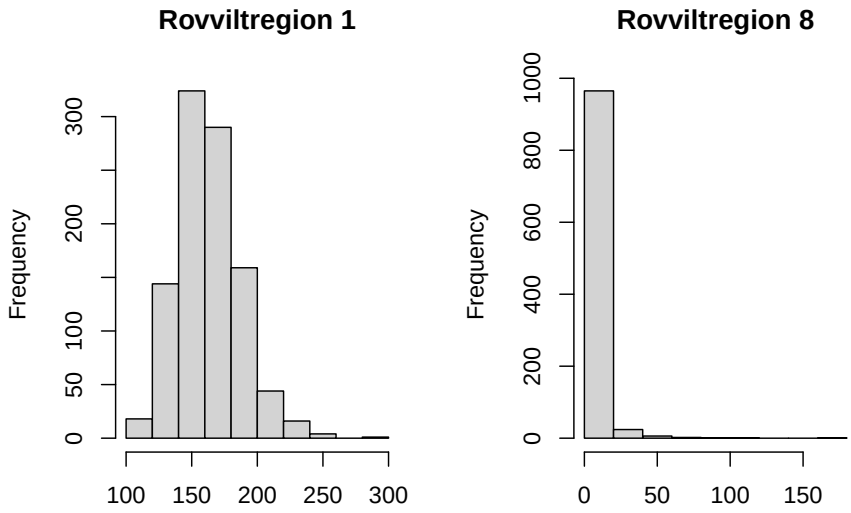
```
par(mar=c(9,5,1,1))
barplot(jerv$indicatorValues$`2019`$expectedValue,
        names.arg = jerv$indicatorValues$`2019`$ICunitName,
        las=2,
        ylab = "Estimated number of\nwolverine in 2019")
```



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 = "")
```

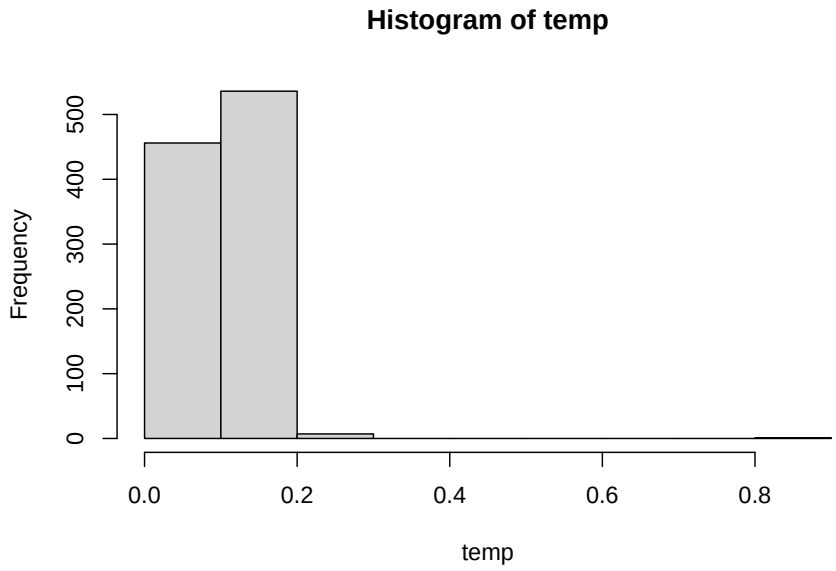


For some reason the extected values are far from the mean of these distributions. I did this exercise once before, and did not get this problem then. I think the difference is that I use eco = NULL this time, in the `importDatasetApi()`, and this cause the output to somehow split into forest and alpine 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
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	163.01	24.19	836.43	0.19	0.15	Rovviltregion 8
1304	59.15	21.66	354.68	0.17	0.37	Rovviltregion 7
1305	13.49	25.35	399.57	0.03	1.88	Rovviltregion 6
1311	3.24	9.42	284.7	0.01	2.91	Rovviltregion 1
1313	3.65	8.94	203.57	0.02	2.45	Rovviltregion 2
1315	9.28	48.24	194.36	0.05	5.2	Rovviltregion 3
1316	3.55	10.38	165.55	0.02	2.92	Rovviltregion 5
5141	3.87	10.09	28.8	0.13	2.61	Rovviltregion 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)
```

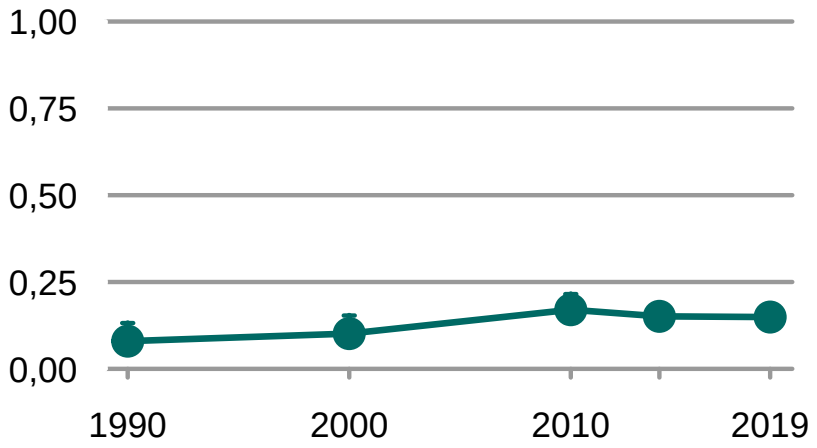


Figure 4.1: The scaled indicator values for wolverine across Norway.

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

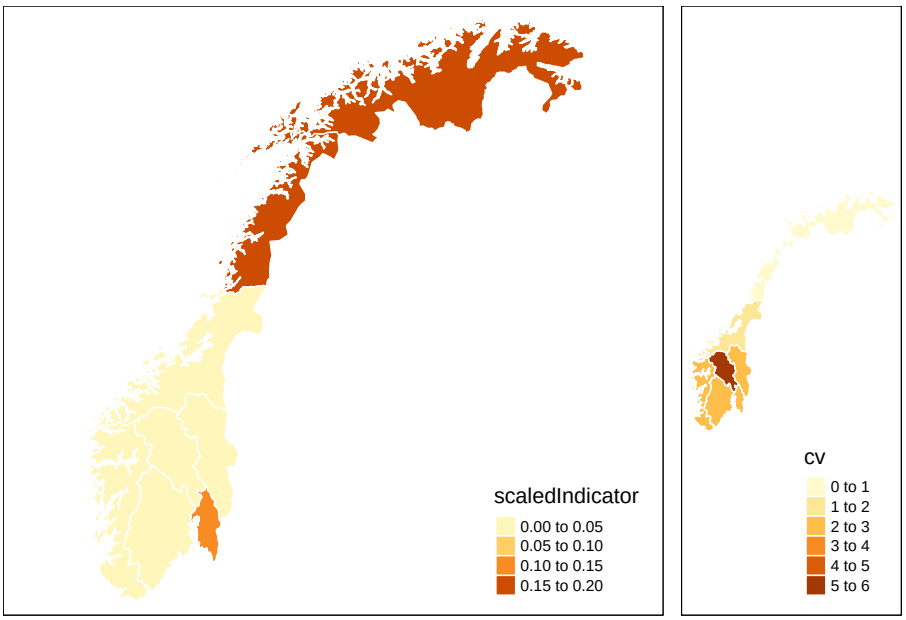
```
path <- "data/outlineOfNorway_EPSG25833.shp"
nor <- sf::read_sf(path)
nor <- st_transform(nor, crs=st_crs(rov))
```

```
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))
```


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 Wordcloud figures

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

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(Pressu

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

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

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

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

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

5.2 The most important pressure factors for dikesoldogg (oblong-leaved sundew)

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

5.3 The most important pressure factors for elg (Moose)

```
wordcloud(words = elg$PressureFactor, freq = elg$PressureValue, min.freq = 1, max.words = 200, ra
```

5.4 The most important pressure factors for havørn (White-tailed eagle)

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

5.5 The most important pressure factors for Lange (Common ling)

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

5.6 The most important pressure factors for Lomvi (common guillemot)

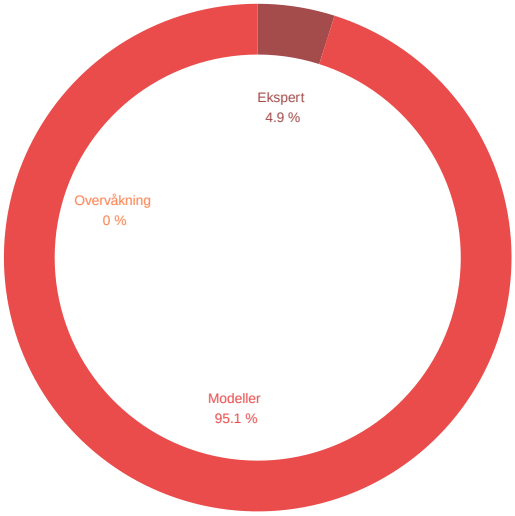
```
wordcloud(words = lomvi$PressureFactor, freq = lomvi$PressureValue, min.freq = 1, max.words = 200
```

5.7 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)
#> -- Attaching packages ------------------- tidyverse 1.3.1 --
#> v ggplot2 3.3.5     v purrr   0.3.4
#> v tibble  3.1.6     v stringr 1.4.0
#> v tidyr   1.2.0     v forcats 0.5.1
#> v readr   2.1.2
#> -- Conflicts ---------------------- tidyverse_conflicts() --
#> x ggplot2::annotate() masks NLP::annotate()
#> x dplyr::filter()     masks stats::filter()
```

```
#> x dplyr::lag()        masks stats::lag()
# 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 here controls
  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")
```


Ekspert
4.9 %
Overvåkning
0 %
Modeller
95.1 %