Species diversity models

Bird species richness

Prof. Dr. Loic Pellissier and Prof. Dr. Niklaus Zimmermann

October 21, 2016

Contents

1 Introduction

The goal of this practical is to learn the use of geographic information systems (GIS) in R . We will prepare a set of environmental predictors that are relevant to explain the richness of birds in the landscape.

We will use the *monir* data set. Monitoring of common breeding birds (Monir by its initials in french) is a central project for the monitoring of common species. It shows the development of their workforce and potential changes to their range.

The field of study consists of 267 square kilometer evenly distributed throughout Switzerland. They come largely from the monitoring network of biodiversity in Switzerland with whom we work closely. Numbers are identified by a simplified method for mapping the territories. 200 members of staff, volunteers in majority, conduct the field work. To process a large number of surfaces, it is necessary that the number of censuses by area is not too high (3 visits per spring 2 above the limit of the forest). The readings always follow the same route and observers note the contacts with the criteria set for each species.

Finally, we will combine environmental factors and species richness in a statistical model to map the diversity of bird species over Switzerland.

We change the working directory to our reference folder (files will be loaded with respect to this address):

```
setwd("/Volumes/Local/emilianodiazsalasp/Documents/ETH Zurich/USYS/scripts/birdRichness/R")
```

Geospatial data are usually only useful when associated with a coordinate reference system (CRS)[1]. This is the information needed to relate spatial coordinates in the dataset to actual positions on our planet, and by extension to relate spatial datasets to one another. When using programmatic tools to convert spatial data into a different CRS (e.g., when performing map projection) you will sometimes need to express it in a standard format that the program can understand. There are several common ways to do this. Different software applications have traditionally used different formats, but most modern software libraries (including those associated with R) will support at least one of the following: EPSG codes, PROJ.4 strings, and the Well-known text (WKT) format.

PROJ.4 strings are a compact way to identify a coordinate reference system. Using the PROJ.4 syntax, it is possible to specify the complete set of parameters that define a particular CRS. The PROJ.4 string is a sequence of parameters, each one beginning with a "+" symbol. The mandatory +proj parameter gives the general projection using a PROJ.4-specific name (note: for unprojected data, use +proj=latlon). If a standard ellipsoid (+ellps) is used, it can be given by name, as can the datum (+datum). You will then need to add any additional parameters (and their values) required to define the specific projection. For many common projections, refer to this online documentation of the relevant parameters. More generally, the PROJ.4 string will be built from a small set of parameters common to most projections.

We will work with the Swiss coordinate reference system CH 1903 LV95 (EPSG 2056) which is defined by the following parameters:

Parameter name	Proj4 name	CH 1903 LV95 value	Proj4 name
Projection name	proj	Oblique mercator	omerc
latitude of origin	lat_0	46' 57' 8.67"	46.94
longitude of origin	lonc	7' 26' 22.5"	7.44
Azimuth	alpha	90	90
Scaling factor	k_0	1	1
False easting	x_0	600,000m	600000
False northing	y_0	200,000m	200000
Units	units	meters	meters
Ellipsoid name	ellps	Bessel	bessel

Table 1: Swiss coordinate reference system (CH 1903 LV95) parameters

For more information on this coordinate system visit the Swiss Federal Office of Topography (swisstopo).

We store CH 1903 LV95 coordinate reference system parameters in a proj.4 string:

```
prj <- "+init=epsg:21781"
```

We will obtain a shape file including the cantons of Switzerland to be used as a reference with which to contextualize other spatial features of Switzerland which we will use in order to explain bird species richness.

Shape objects in R are defined by the *SpatialPoints*, *SpatialLines* and *SpatialPolygons* classes of the `sp` package:

[1] Information in introduction relating to coordinate reference systems (CRS) and PROJ.4 CRS specification format taken from https://www.nceas.ucsb.edu/scicomp/recipes/projections

- SpatialPoints consist of a matrix with n rows and 2 columns, one for each coordinate, where n is the number of points and a string indicating the coordinate reference system in which coordinates of points are expressed,
- SpatialLines consist of a list of *Lines* objects and a string indicating the coordinate reference system in which lines are expressed. Each *Lines* object in the list itself consists of a list of *Line* objects and an identifier, *ID*, for that list. The *Line* object, similar to SpatialPoints, consists of a matrix with n rows and 2 columns.
- SpatialPolygons consist of a list of *Polygons* objects and a string indicating the coordinate reference system in which polygons are expressed. Each *Polygons* object in the list itself consists of a list of *Polygon* objects and and identifier, *ID* for that list. The *Polygon* object, similar to SpatialLines, consists of a matrix with n rows and 2 columns, except that in this case the coordinates of the last row must be the same as that of the first row.

The corresponding *SpatialPointsDataFrame*, *SpatialLinesDataFrame* and *SpatialPolygonsDataFrame* classes allow to store shape objects together with a dataframe where the number of rows in the dataframe corresponds to the number of points (number of rows in coordinate matrix), lines (size of list of *Lines*) or polygons (size of list of *Polygons*). This allows us to associate a vector of variables (a row of the dataframe) to each shape object.

We use the `getData` function from the R package `raster` to get the political map of Switzerland at the level of cantons. This function can be used to get geographic data for anywhere in the world. Data are read from files that are first downloaded if necessary. This data will come expressed with respect to its own coordinate reference system. To use it together with the data we will import, we transform it into the Swiss coordinate system using the `spTransform` function from the R package `sp`.

We get a map of the cantons of Switzerland:

```
library(raster)
print(CRS(prj))
```

```
## CRS arguments:
##  +init=epsg:21781 +proj=somerc +lat_0=46.95240555555556
## +lon_0=7.439583333333333 +k_0=1 +x_0=600000 +y_0=200000
## +ellps=bessel +towgs84=674.4,15.1,405.3,0,0,0,0 +units=m +no_defs
```

```
ch.sp <- getData(c("GADM", "countries", "SRTM", "alt", "worldclim")[1],
    country = "CHE", level = 1)
library(sp)
ch.sp <- spTransform(ch.sp, CRS(prj))
```

Now lets check the class of ch.sp:

```
class(ch.sp)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

What does the SpatialPolygonsDataFrame consist of?

```
slotNames(ch.sp)
```

```
## [1] "data"        "polygons"    "plotOrder"   "bbox"        "proj4string"
```

What is the internal variable polygons?

```
class(ch.sp@polygons)
```

```
## [1] "list"
```

```
length(ch.sp@polygons)
```

```
## [1] 26
```

```
table(sapply(ch.sp@polygons, class))
```

```
##
## Polygons
##       26
```

```
table(sapply(ch.sp@polygons, length))
```

```
##
##  1
## 26
```

```
slotNames(ch.sp@polygons[[1]])
```

```
## [1] "Polygons"  "plotOrder" "labpt"     "ID"        "area"
```

We can see that it is a list of size 26 where each element is a *Polygons* of length 1, which contains, among other internal variables, *Polygons*.

What is the internal variable Polygons?

```
class(ch.sp@polygons[[1]]@Polygons)
```

```
## [1] "list"
```

```
length(ch.sp@polygons[[1]]@Polygons)
```

```
## [1] 1
```

```
slotNames(ch.sp@polygons[[1]]@Polygons[[1]])
```

```
## [1] "labpt"   "area"    "hole"    "ringDir" "coords"
```

```
head(ch.sp@polygons[[1]]@Polygons[[1]]@coords)
```

```
##          [,1]     [,2]
## [1,] 659249.6 273009.8
## [2,] 659257.5 273006.9
## [3,] 659327.2 273009.7
## [4,] 659402.9 273022.4
## [5,] 659458.7 273053.1
## [6,] 659498.6 273086.6
```

```
tail(ch.sp@polygons[[1]]@Polygons[[1]]@coords)
```

```
##             [,1]     [,2]
## [2009,] 659020.0 273378.2
## [2010,] 659057.4 273239.9
## [2011,] 659076.4 273164.6
## [2012,] 659130.7 273077.8
## [2013,] 659186.6 273032.5
## [2014,] 659249.6 273009.8
```

We can see that it is a list of size 1 where each element is a *Polygon*, which contains, among other internal variables, *coords*, a matrix of coordinates describing the points of the polygon and such that the first point is the same as the last.

What is the area of each one of those *Polygon* objects in the nested lists of size 1?

```
format(round(sapply(ch.sp@polygons, function(polys) polys@Polygons[[1]]@area)),
    trim = T, scientific = F, big.mark = ",")
```

```
##  [1] "1,377,235,713" "245,657,490"   "157,914,676"   "4,517,435"
##  [5] "34,714,024"    "970,034"       "5,583,456"     "655,202"
##  [9] "692,289,399"   "71,242"        "832,717,053"   "3,154,504"
## [13] "811,268,715"   "16,301,185"    "73,651,011"    "2,414,966,888"
## [17] "28,557,067"    "889,691,068"   "1,625,231"     "2,922"
## [21] "2,854,637,576" "1,070,768,440" "5,209,024,926" "3,132,996,310"
## [25] "1,738,107,053" "240,370,831"
```

For more information on the shape objects in R go to http://www.maths.lancs.ac.uk/~rowlings/Teaching/UseR2012/cheatsheet.html.

We now take a look at the bird species richness data.

We load and plot bird richness response variable from monir dataset:

```
bird <- read.table("../data/textFiles/Richness_bird.txt", h = T)
par(mar = c(0, 0, 0, 0))
plot(ch.sp)
points(bird$X, bird$Y, cex = bird$richness/50, pch = 21, bg = "yellow")
```

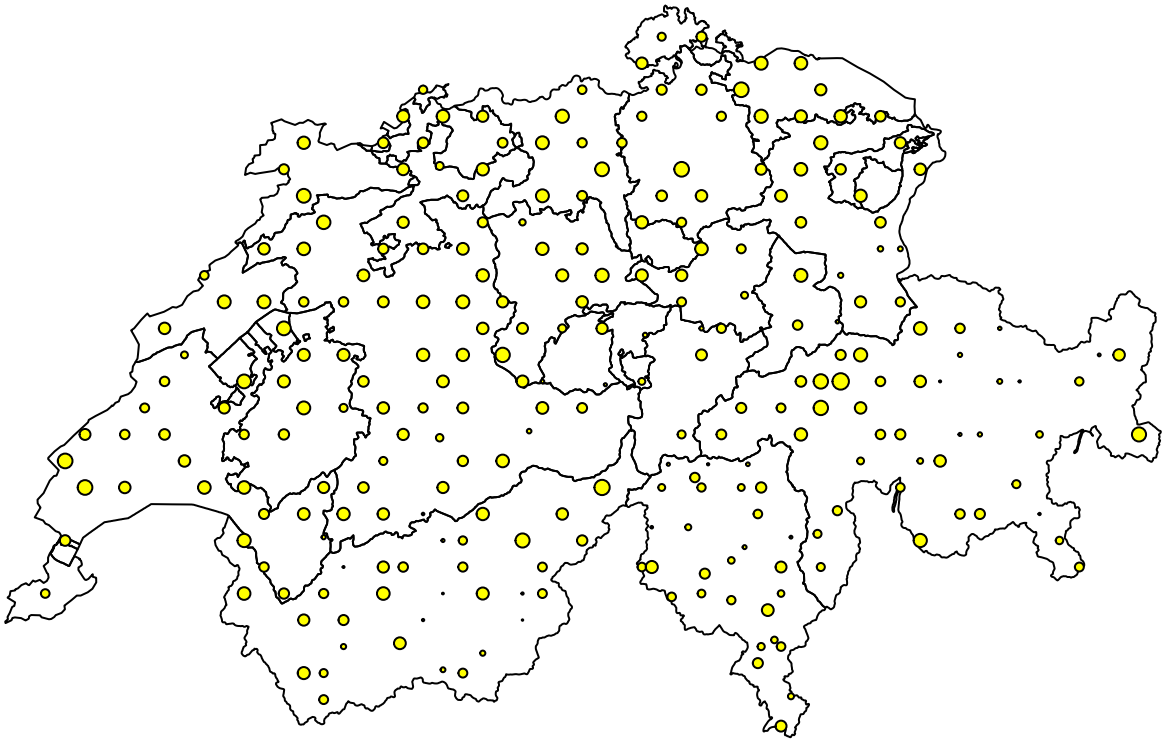

Figure 1: Political map of Switzerland and bird richness at sampled sites (from monir dataset).

Question 1.0.1 *What environmental factors explain bird species richness?*

2 Creation of environmental predictors

We will prepare a set of environmental variables that are expected to explain bird species richness in Switzerland.

A raster object consists primarily of:

- A grid of cells,
- A coordinate reference system (CRS) for the grid and its cells so that we know the location to which the grid refers,
- A variable of interest for which each cell in the grid has a value, and,
- Other information relating to the CRS, projection, resolution, etc.

The strategy will be to obtain predictor rasters for a 1000m x 1000m resolution covering all of Switzerland. To build the model we will then *extract* from each raster the values of the cells where bird species richness was sampled.

The `extract` function from the R package `raster` gives back the values of the raster, given as first argument, at the coordinates given as second argument: in other words the values of the cells where the coordinates *fall*.

2.1 Elevation based predictors

2.1.1 Elevation

We will use elevation, primarily, as a predictor for modeling temperature and slope which are two environmental factors which affect bird species richness.

We will first load a digital elevation model (DEM) raster using the `raster` function from the R package `raster` which creates a raster object in one of several ways, including by reading a file that can be interpreted by the *gdal* driver, which is how we use it here.

The elevation raster was obtained from the Swiss Federal Office of Topography (swisstopo).

We load and plot elevation raster:

```
dem <- raster("../data/mnt25_1km", crs = prj)
plot(dem)
plot(ch.sp, add = T)
```

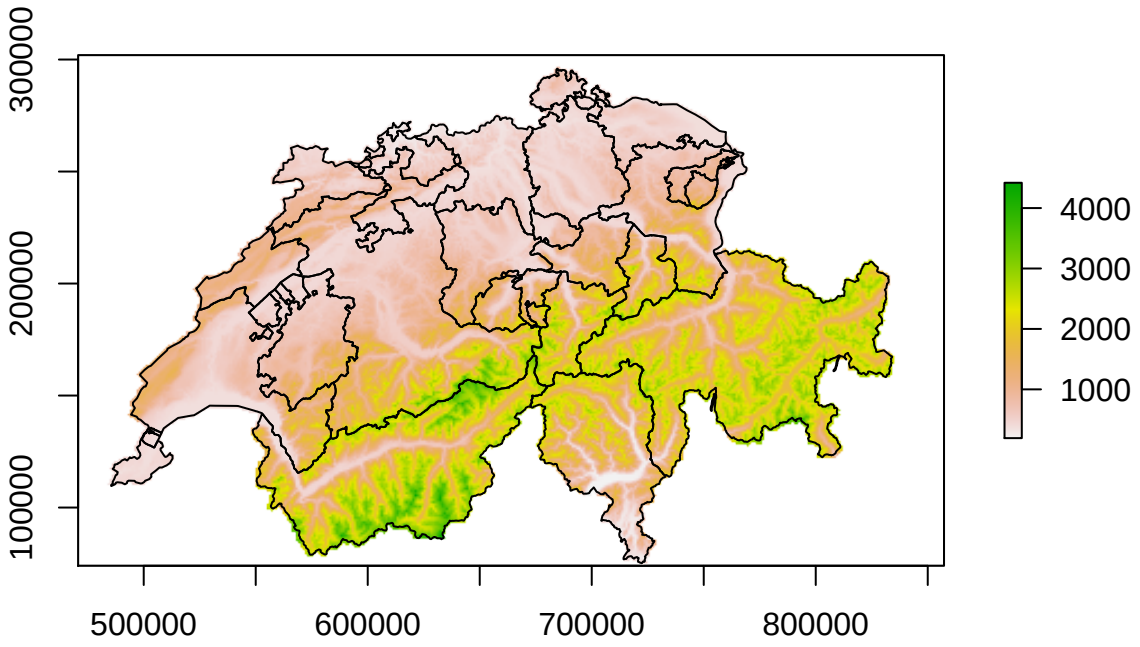

Figure 2: Map of elevation (raster) in Switzerland together with the cantonal borders (shapefile).

2.1.2 Slope

Slope is a proxy of many environmental parameters. In particular, slope is a good proxy of land-use intensity, since steep slopes are more difficult to be worked by farmers.

We will calculate slope using the method from the `terrain` function which is part of the `raster` package. The function calculates slope as a function of elevation so we input the elevation raster as the first argument. This function can calculate several variables so we specify, through the *opt* and *unit* parameters that we need slope in degrees.

We calculate slope using terrain function and plot resulting raster.

```
slope <- terrain(dem, opt = "slope", unit = "degrees")
par(mar = c(0, 0, 0, 0))
plot(slope, axes = F, box = F, col = terrain.colors(100))
```

Figure 3: Map of slope in degrees (raster) in Switzerland calculated from elevation map with terrain function.

2.1.3 Slope categories

The `reclassify` function from the R package `raster` can be used to classify numeric rasters into intervals. To do so provide a three column *reclassification* matrix. The third column indicates the new value assigned to all values that are in the range indicated by the first two columns. If there are overlapping ranges the value corresponding to the range which appears first in the reclassification matrix takes precedence. The parameter *right* which takes values true or false indicates whether the intervals should be closed on the right and open on the left (true) or vice versa (false).

The `values` function from the R package `raster` gives back the values of the raster grid in the order of the cell numbers which go from the top left of the raster grid, accross and then down, to the bottom right of the raster grid.

First categorize slope raster according to the following intervals: 0-10, 10-20, 20-30 and 30-40:

```
summary(values(slope))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    2.58    7.32    9.28   14.73   39.43   42804
```

```
cats <- seq(0, 30, by = 10)
catLabels <- paste(cats, cats + 10, sep = "-")
rcl <- matrix(c(cats, cats + 10), length(cats), 2)
rcl <- cbind(rcl, seq(length(cats)))
slopeCat <- reclassify(slope, rcl, right = F)
```

Now plot newly created slope category raster:

```
library(RColorBrewer)
par(mar = c(0, 0, 0, 0))
plot(slopeCat, legend = F, col = brewer.pal(4, "Accent"), box = F,
    axes = F)
legend("bottomright", legend = catLabels, fill = brewer.pal(4,
    "Accent"))
```

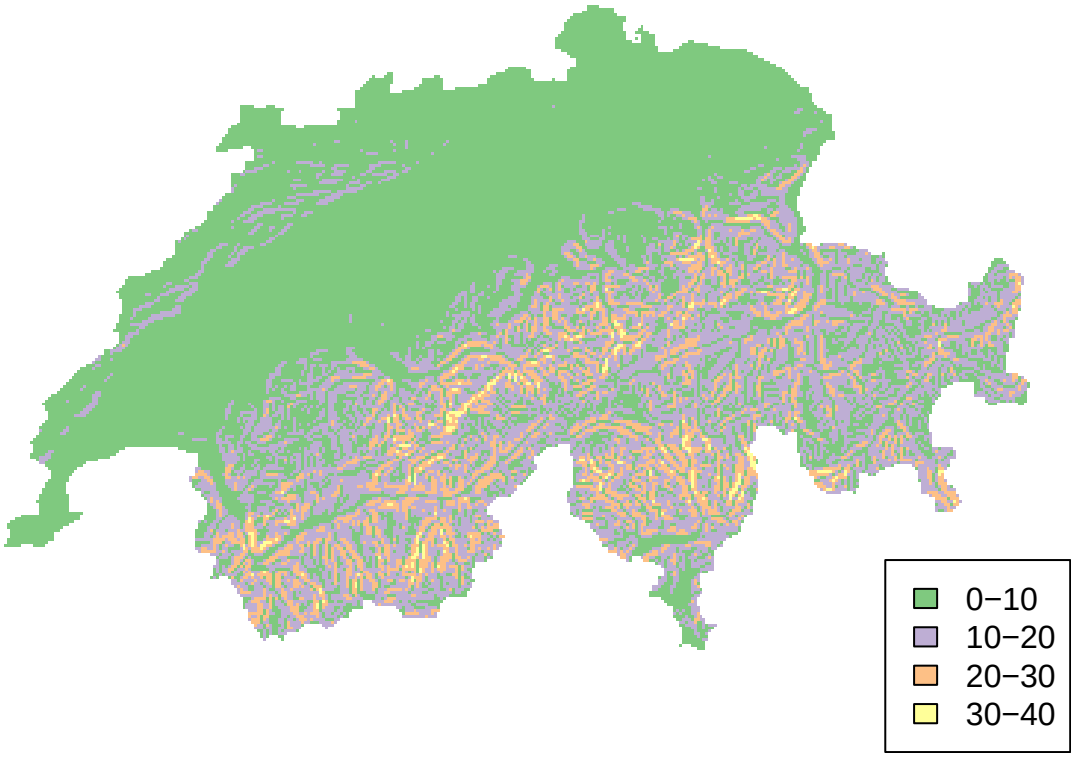

Figure 4: Map of slope in degrees (raster) categorized into four intervals with reclassify function.

2.2 Climate predictors

2.2.1 Temperature estimation

Temperature is a major physiological limitation for many different species including birds. For instance only a few specialized species have the adaptations necessary to tolerate the cold temperatures observed at higher elevation. We want to use mean temperature as a predictor of bird species richness but only have it available at the locations of Swiss meteorological stations. We will estimate a model for temperature that will allow us to estimate mean temperature at any location in Switzerland, including those for which we have bird species richness observations.

We first load the temperature data from Swiss meteorological stations and display a sample of it:

```
library(pander)  # pander
t.obs <- read.table("../data/textFiles/Table_mean.txt", h = T)
pander(head(t.obs[, c(1:5, 33:34)]), caption = "Sample of temperature (yearly average of mean daily temp
```

Station	X	Y	X1980	X1981	X2009	X2010
ABO	609400	148975	8.713	NA	10.67	9.234
AIG	560400	130713	NA	13.61	15.44	14.1
ALT	690174	193558	12.22	12.98	14.63	13.45
AND	752687	164035	NA	NA	13.11	11.81
ARH	760350	261380	NA	NA	14.2	13.09
ATT	586850	105310	NA	NA	1.069	-0.1925

Table 2: Sample of temperature (yearly average of mean daily temperature in degrees celsius) information for first and last two years available of the 1980-2010 period.

The data includes information from 108 meteorological stations from the Swiss meteorological network (meteoswiss). It consists of yearly averages of mean daily temperature in degrees celsius for the 1980-2010 period.

Now plot location of meteorological stations and bird species richness observations:

```
par(mar = c(0, 0, 0, 0))
plot(dem, box = F, axes = F)
points(t.obs[, c("X", "Y")], col = "blue", cex = 1)
points(bird[, c("X", "Y")], pch = 4, col = "dark green")
legend(730000, 90000, col = c("blue", "dark green"), legend = c("meteo-station",
    "bird richness obs."), pch = c(1, 4))
```

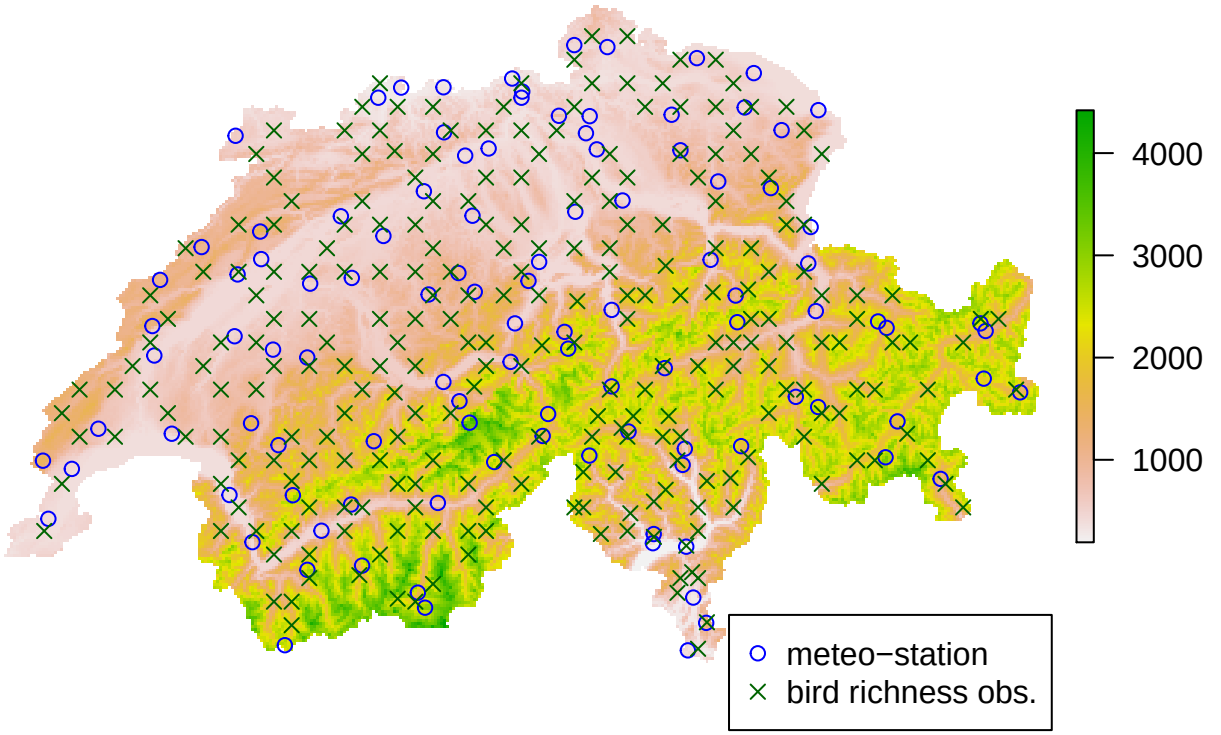

Figure 5: Map of elevation (raster) with locations of meteorological stations and bird species richness observations.

We can see that the stations are reasonably evenly distributed accross Switzerland. We can also observe that we need a model for mean temperature in order to estimate it at bird species richness observation locations and use it as a predictor.

It is well known that mean temperature depends on, among other things, altitude. We will now explore the relationship between mean temperature and elevation.

We compute the mean temperature at all meteorological stations across years 1980-2010:

```
t.mean <- rowMeans(t.obs[, 4:ncol(t.obs)], na.rm = T)
```

Now extract elevation values for the sites with temperature data:

```
t.elev <- extract(dem, t.obs[, c("X", "Y")])
```

Next plot the relationship between elevation and temperature:

```
plot(t.elev, t.mean, xlab = "elevation", ylab = expression("Temperature (" ~
    degree ~ C ~ ")"))
```

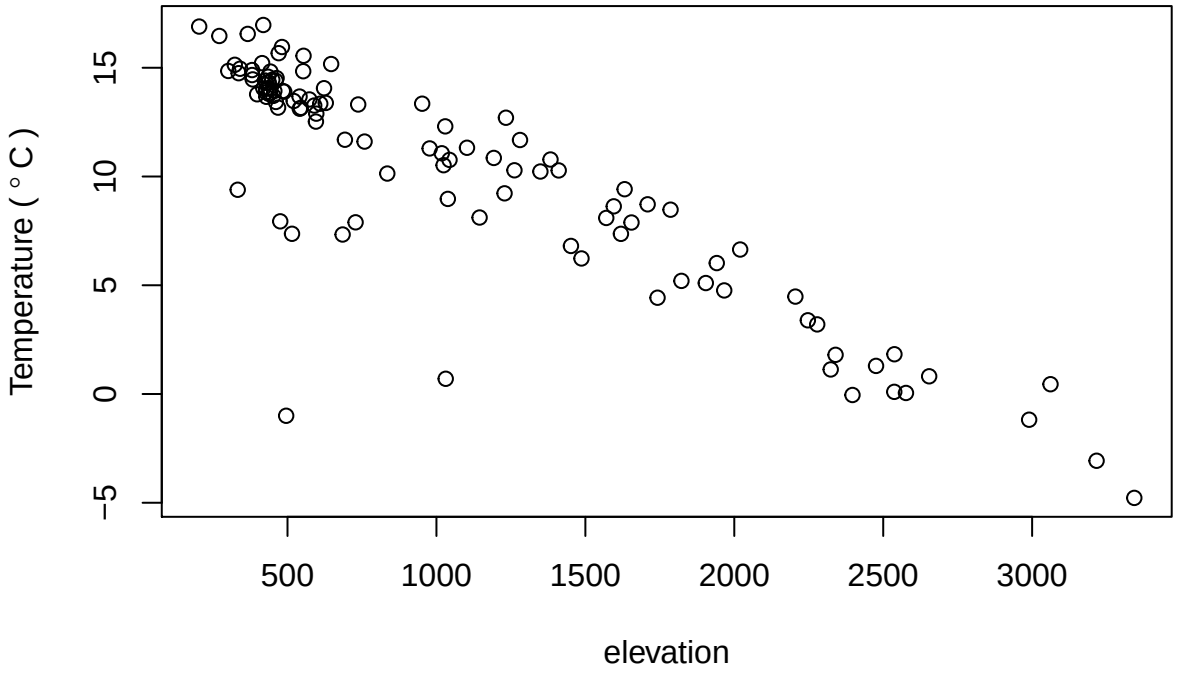

Figure 6: Mean temperature (1980-2010) vs. elevation at meteorological stations in Switzerland.

We observe a negative linear relationship between mean temperature and elevation. Since we have the elevation information for all of Switzerland at 1000m x 1000m resolution, we can use this relationship to build a model of temperature based on elevation, and use this model to predict the mean temperature for all of Switzerland at the given resolution.

We will fit a simple linear model of the type:

$$t_i = \beta_0 + \beta_1 e_i + \epsilon_i \tag{1}$$

where:

- t_i is temperature of observation i,
- e_i is elevation of observation i,
- ϵ_i is the error term corresponding to observation i. We assume the error terms ϵ_i:
  - have mean zero: $\mathbb{E}[\epsilon_i] = 0$ for all i,
  - are homoscedastic: $Var[\epsilon_i] = \sigma^2$ for all i,
  - are not autocorrelated: $Cor[\epsilon_i, \epsilon_j] = 0$ for all i, j, and
  - are distributed normally (not strictly necessary, but standard confidence intervals and hypothesis tests rely on this).

We fit a linear model for temperature based on elevation:

```
lm1 <- lm(t.mean ~ t.elev)
pander(summary(lm1), caption = "Summary of linear model of temperature based on elevation.")
```

	Estimate	Std. Error	t value	Pr(>|t|)
t.elev	-0.005826	0.0002953	-19.73	2.838e-37
(Intercept)	16.41	0.3903	42.03	6.375e-68

Observations	Residual Std. Error	R^2	Adjusted R^2
108	2.404	0.786	0.784

Table 4: Summary of linear model of temperature based on elevation.

Question 2.2.1 *Can you explain what is a lapse rate and to which one of these parameters it corresponds?*

In the linear regression setting we can look at certain plots of the residuals to verify model assumptions and identify outliers:

- The Tukey-Anscombe plot helps check the unbiasedness of the model: $\mathbb{E}[r_i] = 0$
- The Scale-location plot helps check the homoscedasticity of errors: $Var[r_i] = k$
- The normal Q-Q plot helps check that error term is indeed distributed normally
- Cook and leverage plots help identify outliers: observations with atypical values for predictors (high leverage) and which have high influence on model estimation (high Cook's distance).

We now create diagnostic plots to check model assumptions:

```
par(mfrow = c(2, 2))
plot(lm1, 1)
plot(lm1, 2)
plot(lm1, 3)
plot(lm1, 5)
```

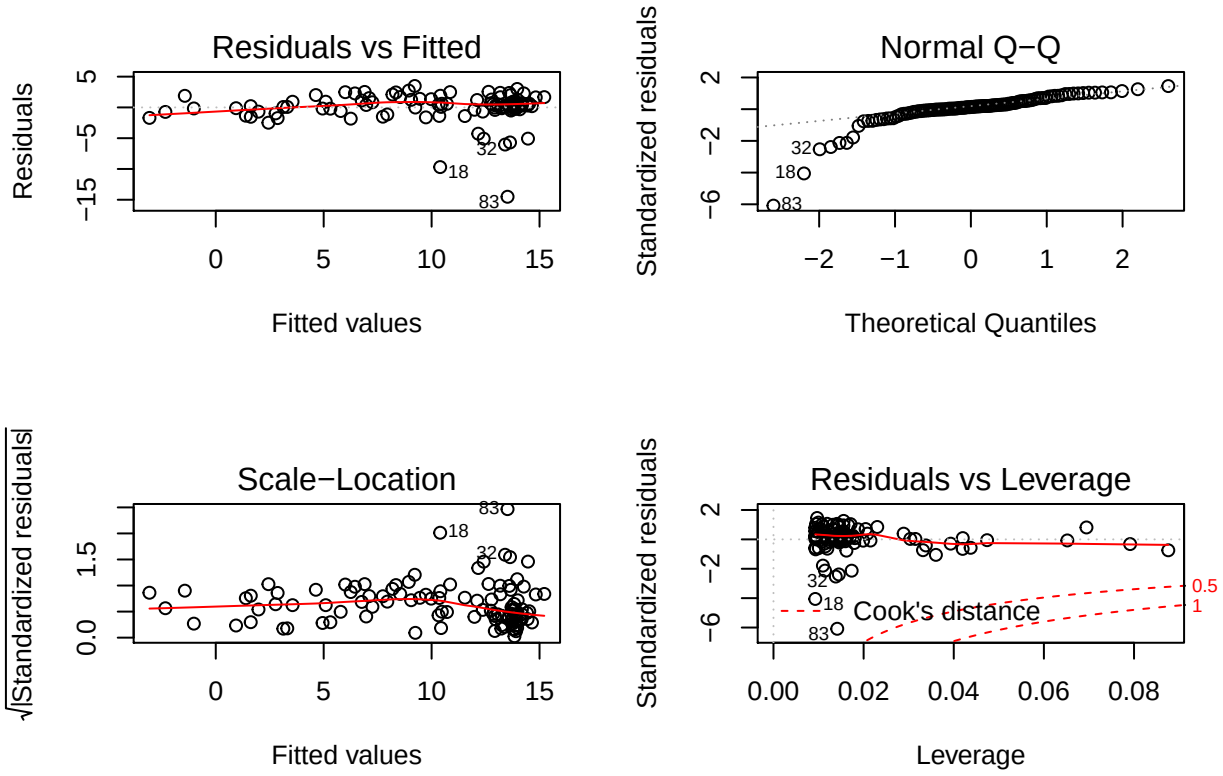

Figure 7: Diagnostics for simple linear regression model of mean temperature as a function of elevation. These plots help to verify model assumptions: Tukey-Anscombe plot (top left) for checking model bias, Normal Q-Q plot (top right) for verifying normality of residuals, Scale-Location plot (bottom left) for verifying homoscedasticity of residuals and residuals vs. leverage plot (bottom right) for detecting outliers.

```
# identify outliers
outliers <- c(18, 32, 83)
```

Model assumptions hold up reasonably apart from observations 18, 32 and 83 which the model does not seem to explain well and which may be affecting parameter estimation. This is because other factors apart from elevation influence temperature. If these other factors are associated to the location of observations we can deal with these outliers by fitting a spatial model for the residuals. This will be our approach.

Lets project model over Switzerland by using coefficients, together with elevation raster of Switzerland, to calculate linear predictor for all locations in Switzerland. We also plot resulting raster:

```
coefs <- coef(lm1)
temperature.r <- dem * coefs[2] + coefs[1]
plot(temperature.r, box = F, axes = F, col = rev(heat.colors(100)))
```

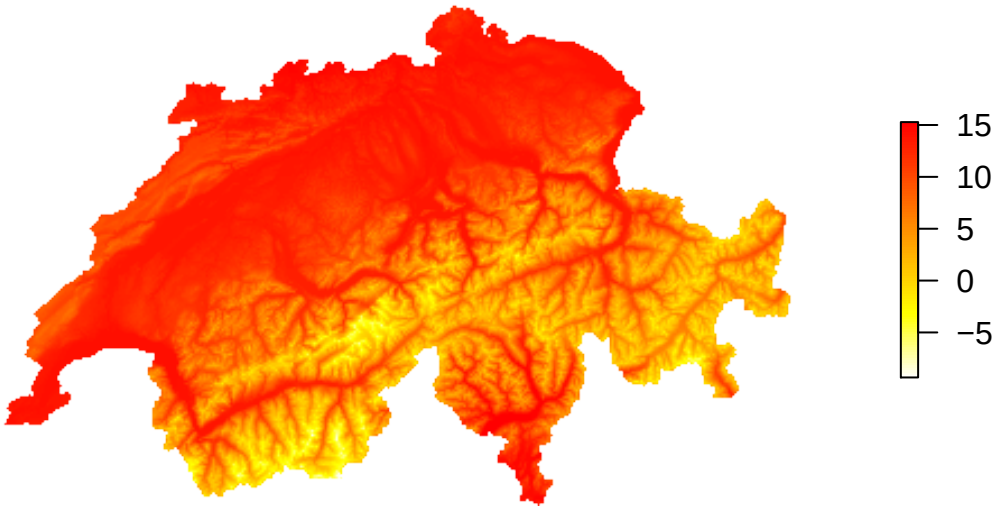

Figure 8: Projected mean temperature of all Switzerland using linear model based on elevation.

For a statistical model to be deemed reasonable it must not have *systematic error*. In practical terms, this means model residuals should not show any pattern or structure when plotted against other variables. In this case, we will plot model residuals against their location to see if there is any spatial pattern in the residuals. Location in itself is not the cause for difference in performance (underestimation or overestimation) of the temperature model based on elevation. Rather location, in this context, will behave as an interaction variable summarizing the effect of all the different factors, except elevation which has already been considered, that affect bird species richness and which have a spatial dependence structure. If there is such a pattern we need to incorporate it into the model so that our prediction of temperature does not have systematic error, in other words, that for any location the expected error is zero.

We only have residuals at certain locations which makes it hard to appreciate any pattern. We first interpolate the residuals so that we obtain a residual value for every location in Switzerland (at 1000m x 1000m resolution).

We first get the residuals from the model and organize them in a dataframe which also contains the coordinates corresponding to each residual:

```
t.obs.res <- data.frame(cbind(t.obs$X, t.obs$Y, residuals(lm1)))
colnames(t.obs.res) <- c("x", "y", "Res")
```

We will now interpolate residuals so as to have a residual value for all cells in the Switzerland 1000m x 1000m raster grid.

The `gstat` function from the R package `gstat` creates a kriging modeling object (kriging is a type of interpolation based on spatial correlation functions) of type gstat. In this case we want to interpolate residuals. The function has the following parameters (among others):

- **formula**: Since we wan to perform ordinary/simple kriging we use formula residual ~ 1, ie we only use the residuals themselves at known locations to infer residuals at unknown locations,
- **locations**: independent variables, i.e. coordinates, and,
- **degree**: order of trend surface in the location, between 0 and 3.

First fit ordinary kriging model to residuals:

```
library(gstat)  #gstat
idw.t.obs <- gstat(id = "Days.Res", formula = t.obs.res$Res ~
    1, locations = ~x + y, data = t.obs.res, nmax = 10, degree = 1)
```

The `interpolate` function from the R package `raster` uses the interpolation model for a given variable z passed as second argument to predict, or interpolate, the values of z for all cells of the raster passed as first argument.

Now interpolate residuals at all locations of 1000m x 1000m Switzerland raster grid:

```
idw.final <- interpolate(dem, idw.t.obs)
```

```
## [ordinary or weighted least squares prediction]
```

The residual values were interpolated at locations of all cells of the elevation raster, even those for which the elevation value is NA. We only need to keep the interpolated residuals for those cells for which the elevation values exists. The function `mask` from the R package `raster` creates a new raster object that has the same values as the first raster argument, except for the cells that are NA in the second raster argument. These cells become NA.

We eliminate interpolated values at places where there is no elevation value:

```
idw.final <- mask(idw.final, dem)
```

Next we plot spatial distribution of residuals based on interpolation of model residuals. We also add locations of meteorological stations and outliers:

```
par(mar = c(0, 0, 0, 0))
plot(idw.final, box = F, axes = F)
plot(ch.sp, add = T)
points(t.obs[outliers, 2:3], col = "blue")
points(t.obs[-outliers, 2:3], col = "black")
text(t.obs[outliers, 2:3] + 10000, labels = outliers, col = "blue")
legend(730000, 90000, col = c("black", "blue"), legen = c("normal observation",
    "outlier"), pch = 1)
```

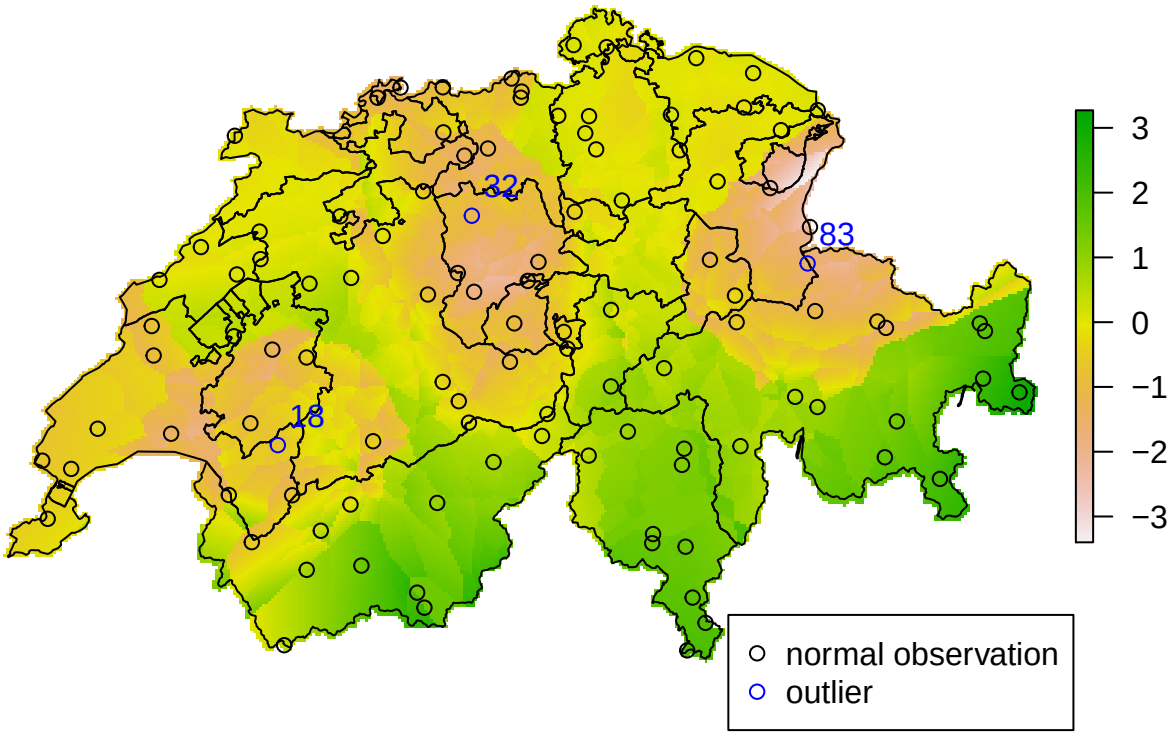

Figure 9: Interpolation model of residuals and observation locations with outliers marked in blue.

Question 2.2.2 *Why is it important to look at model residuals spatially?*

Question 2.2.3 *Can you identify regions with higher residuals (i.e. difference between observation and prediction)? What could explain this?*

Question 2.2.4 *Describe the spatial pattern of errors. Is the climate model biased?*

We can see that the spatial model tends to correct the linear model more near the locations where we observed the outliers indicating that the outliers fit into the spatial pattern of residuals.

We conclude that the linear model for temperature based on elevation makes systematic errors depending on location. Since interpolation model gives us a prediction for the error of our linear model, we can improve this model by adding to it the predicted error based on the interpolation model. Our model is now:

$$t_i = \beta_0 + \beta_1 e_i + \epsilon(x_i, y_i) + \eta_i \tag{2}$$

where:

- t_i is temperature of observation i,
- e_i is elevation of observation i,
- $\epsilon(x_i, y_i)$ is a term that is a function of location, and which represents all predictors, apart from elevation, that depend on location, and,
- η_i is the error term corresponding to observation i. We assume the error terms η_i satisfy the assumptions that previoulsy applied to ϵ_i.

Notice that we fitted the above model in two steps, first estimating the β_0, and β_1 coefficients and then fitted an interpolation model to estimate $\epsilon(x_i, y_i)$. It is also possible (and more optimal in general) to fit all model parameters concurrently.

We correct systematic errors of linear model for mean temperature based on elevation by summing the model and residual projection rasters. We also plot resulting raster:

```
temperature.r <- idw.final + temperature.r
par(mar = c(0, 0, 0, 0))
plot(temperature.r, box = F, axes = F, col = rev(heat.colors(20)))
plot(ch.sp, add = T)
```

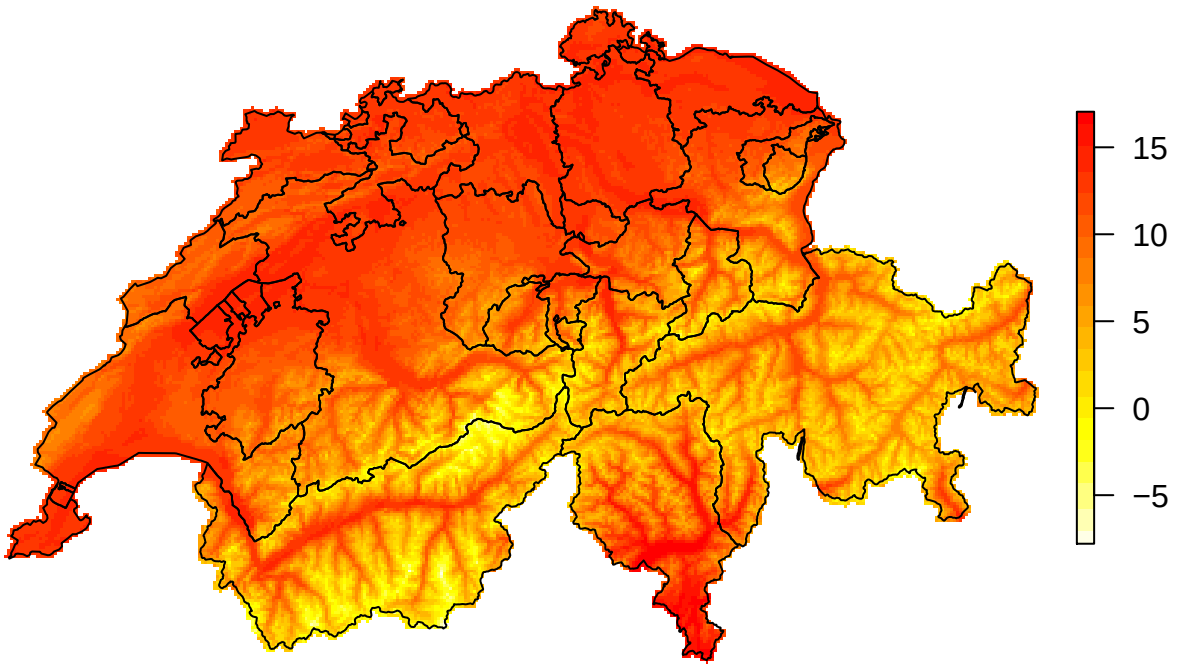

Figure 10: Projected mean temperature of all Switzerland using linear model based on elevation and spatial model of residuals to correct systematic errors.

We now have an expected mean temperature raster at a 1000m x 1000m resolution which we can use to obtain mean temperature at the bird species richness sample locations.

2.2.2 Solar radiation estimation

Solar radiation is another relevant measure of energy, complementary to temperature. This is because temperature measured by meteorological stations corresponds to standardized **shade** temperature.

We wish to estimate the solar radiation for a certain date (June, day 152-181). We will use the function `insolation` from the R package `insol` to achieve this.

We will obtain radiation estimation using `insolation` function and related functions (`JD`, `sunvector`, `sunpos`) from R package `insol`.

We will arbitrarily choose to estimate the radiation on a given 1st of June.

First we obtain the 2016-06-01 in julian-day format:

```
library(insol)  # JD, slope, cgrad, insolation
jd <- JD(as.POSIXct(as.Date("2016-06-01")))
```

Radiation at location x will be estimated as a function of, among other things, the solar zenith angle at x: the angle between the zenith at x and the centre of the sun's disc. The solar zenith angle at x can be calculated as a function of the unit vector in the direction of the sun (from x) , using the `sunpos` function. In turn, using the function `sunvector` the unit vector to the sun at x can be calculated as a function of the julian-day, and the location of x expressed in the non-projected, latitude-longitude, reference system.

To obtain the latitude-longitude coordinates of the raster grid, we obtain the coordinates in the Swiss coordinate system using the function `coordinates` from the R package `sp` and *inverse-project* them, back into the latitude-longitude CRS using the function `project` from the R package `rgdal`. The projection inversion is called by passing the value *TRUE* to the argument *inv*.

Next we btain solar zenith angle:

```
library(rgdal)  # projInfo, project, readOGR
lonLat <- project(coordinates(dem), proj = prj, inv = TRUE)
sunv <- sunvector(jd, lonLat[, 1], lonLat[, 2], 0)
zenith <- sunpos(sunv)[, 2]
```

The `insolation` function estimates radiation as a function of solar zenith angle (*zenith*), julian-day (*jd*), elevation (*height*) and temperature (*tempK*), for which we have an estimate at every relevant location, and, visibility (*visibility*), relative humidity (*RH*), ozone thickness (*03*) and albedo (*alphag*), for which we have no estimate and, in the context of this exercise, set arbitrary but plausible values for these arguments.

Finally we estimate solar radiation and create radiation raster:

```
rad <- insolation(zenith = zenith, jd = jd, height = values(dem),
    visibility = 20, RH = 80, tempK = values(temperature.r) +
        273.15, O3 = 0.002, alphag = 0.15)
radiation <- dem
values(radiation) <- rad[, 1]
```

Now we plot estimated solar radiation raster:

```
par(mar = c(0, 0, 0, 0))
plot(radiation, box = F, axes = F, col = rev(heat.colors(20)))
plot(ch.sp, add = T)
```

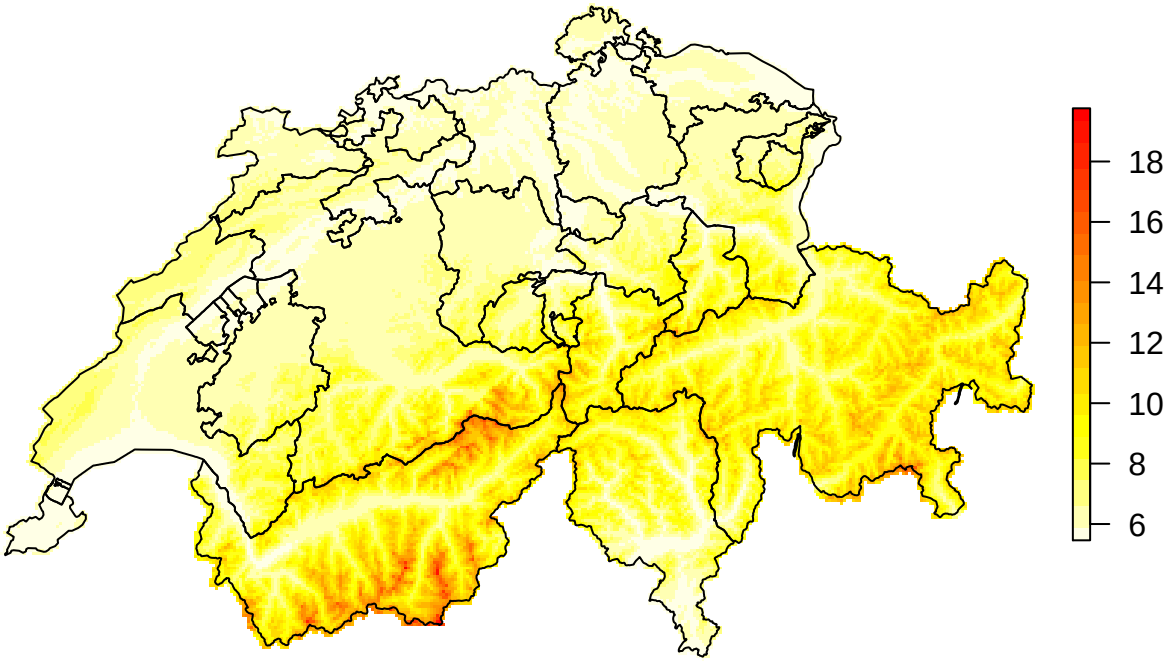

Figure 11: Map of radiation on the 6th of June (raster) for all Switzerland estimated using insolation function.

2.2.3 Moisture index

Birds rely on water to survive so it is likely that a good predictor of bird species richness at a location will be water availability. We will use the moisture index in the month of July, the driest and of greatest significance to species occurrence, as a proxy for water availability.

The monthly moisture index, **mind**, is calculated as monthly precipitation minus monthly potential evapotransporation (PET). Monthly PET was calculated using the Turc formula:

$$PET = \frac{0.4T(R_s + 50)}{T + 15} \tag{3}$$

Where:

- T is monthly mean temperature in degrees celsius, and
- R_s is monthly mean global radiation (cal/cm^2)

Monthly mean global radiation is calculated as:

$$R_s = \frac{(100 - C)R_d}{100} \tag{4}$$

Where:

- C is monthly cloudiness (%), and
- R_d is monthly potential direct solar radiation.

Potential direct solar radiation was calculated by using the empirical formula of Muller (1984) which is valid between 1500 m and 4000 m and with which radiation can be calculated for given sun positions (azimuth and zenith angle) from latitude and elevation. Hourly values of radiation for level surfaces were calculated and corrected for topographic overshadowing, actual slope, and aspect values from the DEM. Daily values were then calculated by integrating the hourly intervals using the Simpson-integral (Press et al. 1989). Because these calculations are very time consuming, monthly totals were derived from a linear interpolation of 10-day intervals.

Monthly moisture indices were calculated for all months of July for which input variables were available at all raster cell locations (1000m x 1000m resolution). Interpolation models for all input variables were first developed so that moisture index could be calculated at all raster cell locations. The July moisture index rasters were then averaged to obtain the moisture index raster which we now load.

We load moisture index raster:

```
MIND <- raster("../data/mind_1km")
```

We now plot the moisture index raster:

```
par(mar = c(0, 0, 0, 0))
plot(MIND, col = brewer.pal(9, "PuBu"), box = F, axes = F, legend = T)
```

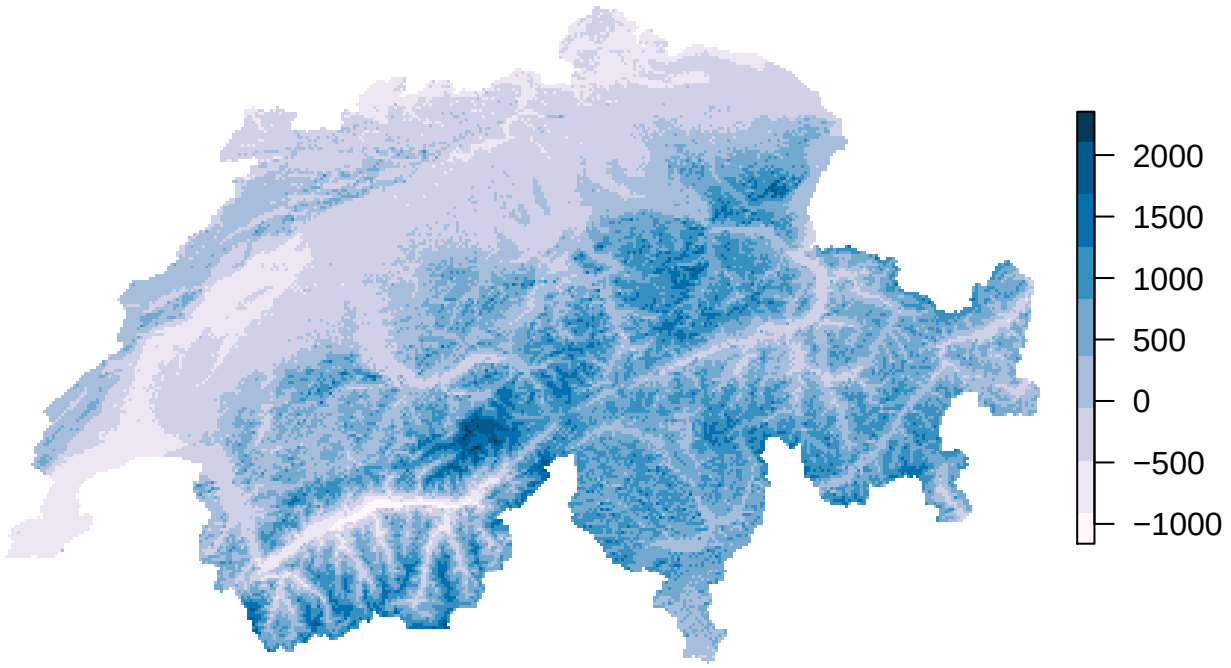

Figure 12: Map of moisture index (raster) for all of Switzerland

Question 2.2.5 *Can you propose other environmental predictors that could influence bird species richness?*

Question 2.2.6 *According to your knowledge of the birdś ecology, which are the best predictors of bird species richness in Switzerland?*

2.3 Landscape predictors

2.3.1 Sum of forest edges

Forest edges provide a diverse and structured habitat favoured by many different species meaning it could be a good predictor of bird species richness. The forest edges raster, which we willnow load, was obtained from the Swiss Federal Office of Topography (swisstopo).

We load and plot forest edges raster:

```
edge <- raster("../data/forest-edges_bl/forest-edges_bl.rst")
# table(values(edge))
par(mar = c(0, 0, 0, 0))
plot(edge, col = c("white", "green4"), axes = F, box = F)
plot(ch.sp, add = T)
```

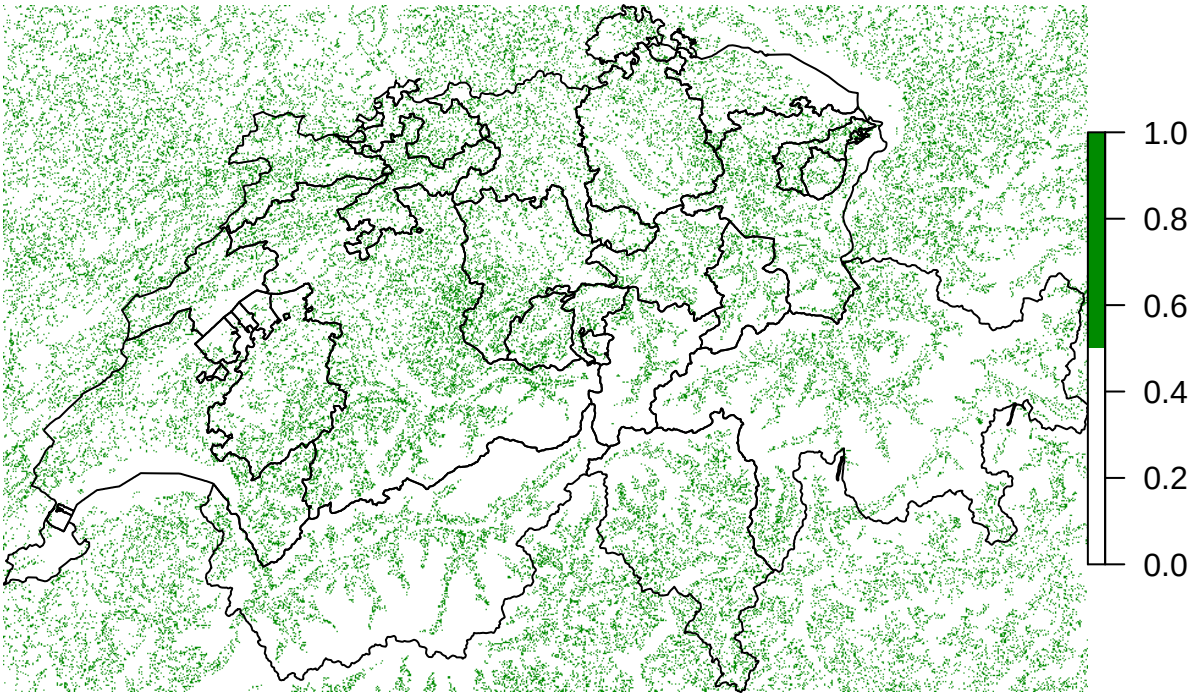

Figure 13: Map of the location of forest edges (raster) over Switzerland.

It is desirable to work with raster layers that have the same grid. A raster grid is uniquely defined by an origin, a point that is one of the intersections of grid lines, and its resolution, the magnitude of each cell-side. In the R package `raster` the origin is defined as the point closest to (0, 0) that is still an intersection of grid lines. The function `origin` and `res` from the R package `raster` return the origin and resolution of a raster object.

We check that the forest edge and elevation rasters share the same grid:

```
origin(edge)
```

```
## [1] 0 0
```

```
origin(dem)
```

```
## [1] -12.5 -12.5
```

```
res(edge)
```

```
## [1] 100 100
```

```
res(dem)
```

```
## [1] 1000 1000
```

We can see that the elevation and forest edge rasters (dem and edge), have different resolutions and are not aligned since they have a different origin. We can visualize this by plotting the raster grids.

We want to zoom in on a part of the raster grids to better visualize them. The *range* of a grid in its x and y coordinates is called its *extent*. The function `extent` from the R package `raster` creates an *extent* object by calculating the extent of a raster object or matrix of coordinates, or, from a vector of four numbers which includes the maximum and minimum x and y values. We will create an square extent of 25 cells by inputing the desired maximum and miniminum x and y values, using the center of the Switzerland raster grid as reference point.

We compute a 5 x 5 cell extent inside elevation (dem) raster. First we locate center of elevation raster:

```
center <- c(x = (dem@extent[1] + dem@extent[2])/2, y = (dem@extent[3] +
    dem@extent[4])/2)
```

Now we use center to create a small extent of 5 x 5 cells corresponding to a square of 5km x 5km:

```
extSmall <- extent(c(center[1] - 500 - 2 * 1000, center[1] +
    500 + 2 * 1000, center[2] - 500 - 2 * 1000, center[2] + 500 +
    2 * 1000))
```

We now wish to obtain the coordinates of the cell centers, together with the corresponding raster values, for those cells that are within the reduced extent. The `crop` function from the R package `raster` returns a geographic subset of the first raster argument as specified by the extent object (or the extent of a raster object) that is the second argument.

We now obtain the cell coordinates for cell centers of both grids within this new extent:

```
cells_edge <- coordinates(crop(edge, extSmall))
cells_dem <- coordinates(crop(dem, extSmall))
```

Now we append raster values for both raster objects that are within this new extent:

```
cells_edge <- cbind(cells_edge, values(crop(edge, extSmall)))
cells_dem <- cbind(cells_dem, values(crop(dem, extSmall)))
```

Next we plot the grids and raster values:

```
plot(cells_edge[, 1:2], pch = 3, xlab = "", ylab = )
lines(cells_dem[, 1:2], type = "p", col = "blue", pch = 3)
text(cells_edge[, 1:2] + 50, labels = cells_edge[, 3], cex = 0.3)
text(cells_dem[, 1:2] + 50, labels = round(cells_dem[, 3]), cex = 1,
    col = "blue")
legend("bottomright", col = c("black", "blue"), legend = c("number of forest edges",
    "elevation"), pch = c(4, 3), text.col = c("black", "blue"),
    bg = "white")
```

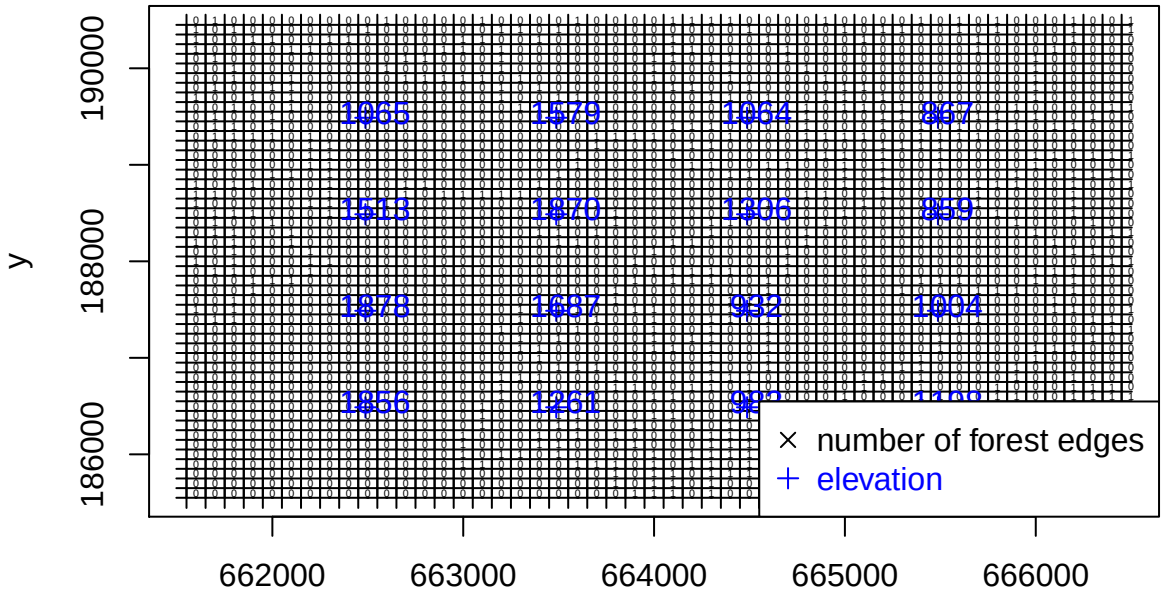

Figure 14: Elevation and forest edges raster grids for 5000m x 5000m extent at the center of entire raster grid.

First we observe that the forest edge raster has a much higher resolution than the elevation raster. We need to transform the forest edge raster so that it has the same grid (same origin and resolution) as the elevation raster permitting us to combine them in our analysis. We can achieve this using the `aggregate` function from the R package `raster` to obtain a sum of edges raster at 1000m x 1000m resolution. The `aggregate` function takes three main arguments:

- **x**: the raster object to aggregate,
- **fact**: aggregation factor, i.e. the number of current cells that will make up the side of one of the new cells, and,
- **fun**: the function to apply to summarize the raster values of the fact^2 (in this case 10^2=100) cells.

In this case we are interested in how many forest edges lie within a 1000m x 1000m cell so we use *sum* as argument value for *fun*.

We aggregate the forest edge raster at 1km:

```
edge.sum <- aggregate(edge, fact = 10, fun = sum)
```

Now we get cell coordinates for cell centers of aggregated grid and append aggregated raster values:

```
cells_edge.sum <- coordinates(crop(edge.sum, extSmall))
cells_edge.sum <- cbind(cells_edge.sum, values(crop(edge.sum,
    extSmall)))
```

Next we plot grid and raster values of aggregated forest edge raster:

```
plot(cells_edge.sum[, 1:2], pch = 3, xlab = "", ylab = "")
lines(cells_dem[, 1:2], type = "p", col = "blue", pch = 3)
text(cells_edge.sum[, 1:2] + 70, labels = round(cells_edge.sum[,
    3]), cex = 0.7)
text(cells_dem[, 1:2] + 70, labels = round(cells_dem[, 3]), cex = 0.9,
    col = "blue")
legend("bottomright", col = c("black", "blue"), legend = c("number of forest edges",
    "elevation"), pch = c(4, 3), text.col = c("black", "blue"),
    bg = "white")
```

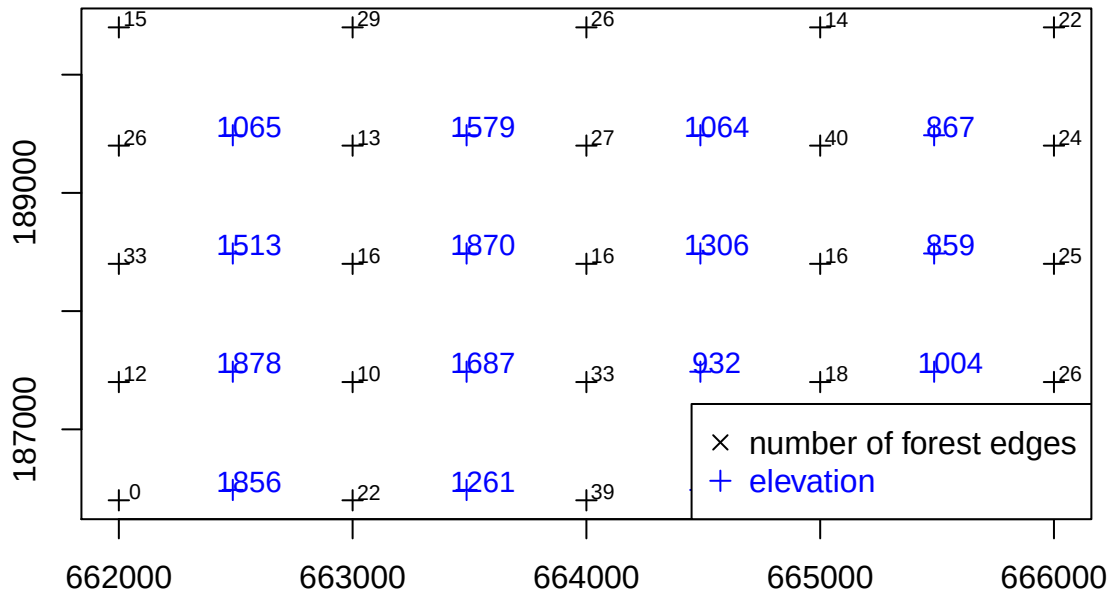

Figure 15: Elevation raster grid and forest edges raster grid aggregated to the same resolution. Zoom in of 5000m x 5000m extent at the center of entire raster grid.

Although, the resolution is now the same, the grids are clearly not aligned. The `resample` function from the R package `raster` will work out the values of the first raster (the one that needs transforming) at the coordinates of the second (our *model* raster) by interpolation and then change the coordinates of the first to those of the second.

Now we align grids by *resampling* grid points:

```
edge.sum <- resample(edge.sum, dem)
```

Next we calculate cell center coordinates of aggregated and aligned forest edge grid and append interpolated raster values:

```
cells_edge.sum <- coordinates(crop(edge.sum, extSmall))
cells_edge.sum <- cbind(cells_edge.sum, values(crop(edge.sum,
    extSmall)))
```

Now we plot grid and raster values of aggregated and aligned forest edge raster:

```
plot(cells_edge.sum[, 1:2], pch = 3, xlab = "", ylab = "")
lines(cells_dem[, 1:2], type = "p", col = "blue", pch = 3)
text(cells_edge.sum[, 1:2] + 70, labels = round(cells_edge.sum[,
    3]), cex = 0.7)
text(cells_dem[, 1:2] - 70, labels = round(cells_dem[, 3]), cex = 0.9,
    col = "blue")
legend("bottomright", col = c("black", "blue"), legend = c("number of forest edges",
    "elevation"), pch = c(4, 3), text.col = c("black", "blue"),
    bg = "white")
```

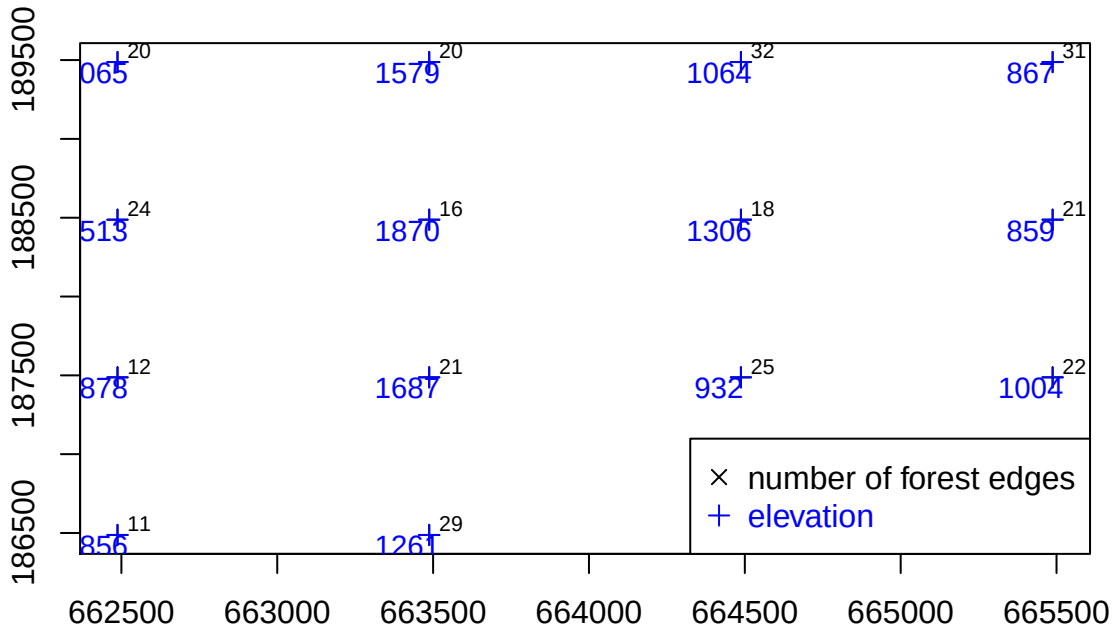

Figure 16: Elevation raster grid and forest edges raster grid aggregated and aligned. Zoom in of 5000m x 5000m extent at the center of entire raster grid.

Both rasters now have the same grid.

Next we will filter the edge sum raster using the mask function to discard raster values that fall outside Switzerland replacing them with NA values. We plot resulting raster:

```
edge.sum <- mask(edge.sum, dem)
par(mar = c(0, 0, 0, 0))
plot(edge.sum, col = brewer.pal(9, "Greens"), axes = F, box = F)
```

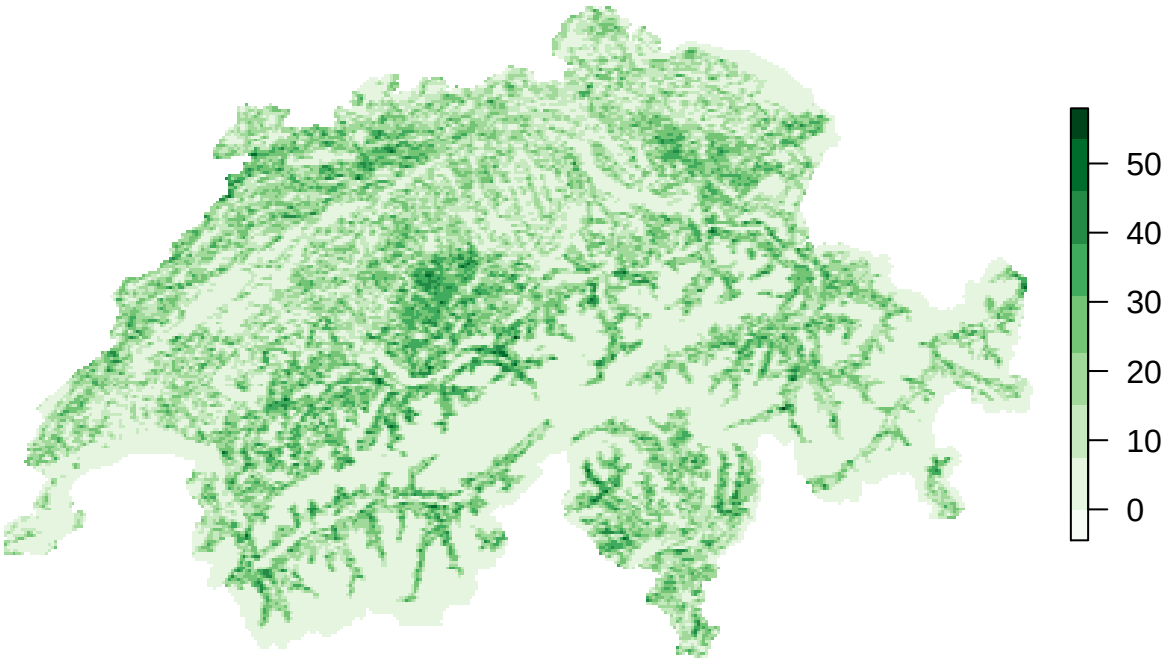

Figure 17: Map of sum of forest edges (raster) in Switzerland. This raster counts the number of 100m x 100m forest edge cells within each 1000m x 1000m cell.

2.3.2 Land-use

We will use land-use information in order to identify the forest habitat that is cruicial for bird species to proliferate and prosper. We will also use this information to construct landcover diversity indicators since birds benefit from a varied habitat.

The land-use categories according to the Swiss Federal Statistics Office' (SFSO) Standard Nomenclature NOAS04 are:

Code	Land-use type
Buildings	
01	industrial and commercial buildings
02	grounds adjacent to industrial and craft buildings
03	single and duplex housing
04	grounds adjacent to single and duplex housing
05	houses in rows and in terraces
06	grounds adjacent to houses in rows and terraces
07	residential buildings
08	grounds adjacent to residential buildings
09	public buildings
10	grounds adjacent to public buildings
11	agricultural buildings ??? whats the difference with 4

Code	Land-use type
12	grounds next to agricultural buildings
13	unspecified buildings
14	grounds next to unspecified buildings
Forest	
50	normal Forest
51	narrow forest
52	reforested area (human reforestation)
53	recently suppressed forest
54	devastated woodland area
55	sparse forest (on agricultural land)
56	sparse forest (on non-agricultural land)
57	low-canopy forest
58	groves, hedges
59	groups of trees (on agricultural land)
60	groups of trees (on non-agricultural land)
Grasslands	
42	natural grassland
43	local pasture
44	overgrown local meadows and pastures
45	mowed pasture
46	favourable pastures
47	overgrown pastures
48	rocky pastures
49	Alp sheep pastures
65	non-agricultural herbaceous vegetation
others	
15-41, 61-64 and >65	other land covers

Table 5: SFSO land use categories according to standard nomenclature NOAS04

We will now load the land-use census of 2009 carried out by the SFSO. Raster values range from 1-72 where each code value represents different land-uses according to above table.

We load and plot land-use map of Switzerland (2009 version):

```
lu09 <- raster("../data/landUse/as09_72.tif")
par(mar = c(0, 0, 0, 0))
plot(lu09, box = F, axes = F, col = rainbow(73))
```

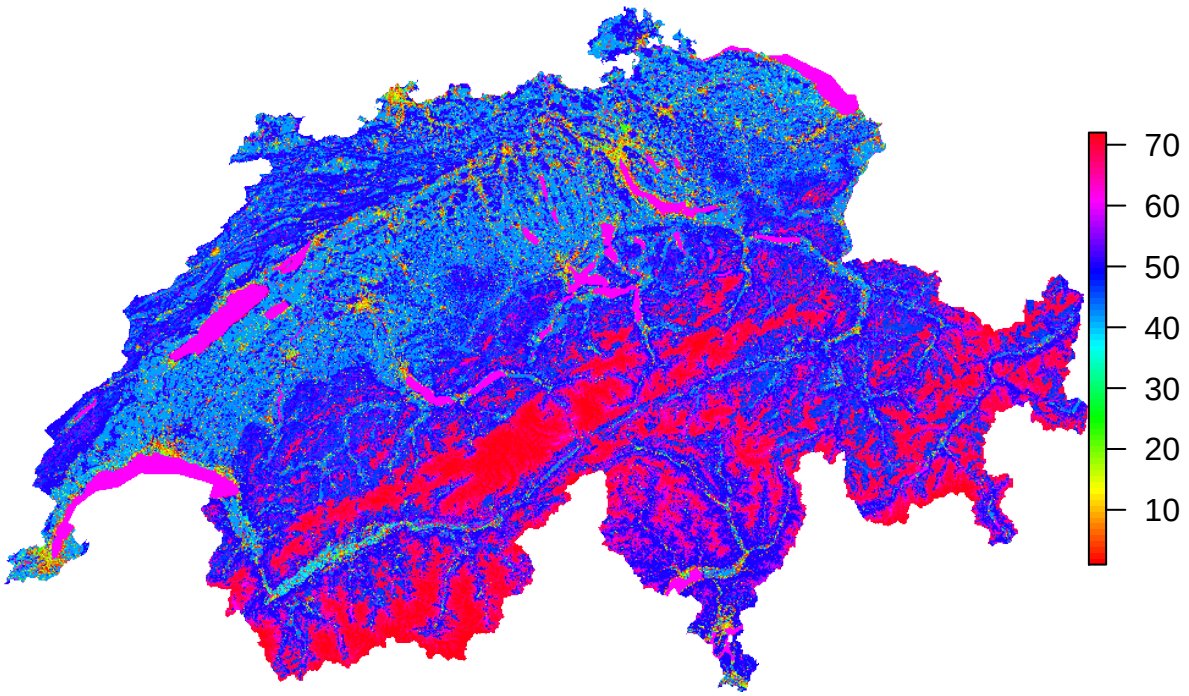

Figure 18: Map of land use in Switzerland in 2009 (raster). Classification made by SFSO according to above table.

2.3.3 Sum of forests

In addition to the sum of forest edges, we will now compute the total surface of forest for each cell of 1000m x 1000m. Forests represent a basic resource of birds for food and and shelter so this could be an important predictor.

2.3.3.1 Land-use classification

According to the Swiss Federal Statistics Office (SFSO) land-use can be divided into 72 categories. According to above table we can classify land-use into 4 super-categories:

- buildings: 1-14
- grassland: 42-49 and 65
- forest: 50-60
- others: 15-41, 61-64 and >65

We will now reclassify the land-use reaster from 72 categories to 4 super categories.

Lets classify land-use type raster according to SFSO super-categories:

```
categories <- c("building", "other", "grassland", "forest", "other")
glu09 <- lu09
values(glu09) <- as.factor(categories[as.numeric(cut(values(lu09),
    breaks = c(0, 14, 41, 49, 60, 72), include.lowest = T))])
values(glu09)[which(values(lu09) == 65)] <- as.factor(categories)[3]
```

Now we plot super-category land-use raster:

```
categories <- levels(as.factor(categories))
par(mar = c(0, 0, 0, 0))
plot(trim(glu09), col = topo.colors(4), legend = F, axes = F,
    box = F)
legend("bottomright", legend = categories, fill = topo.colors(4))
```

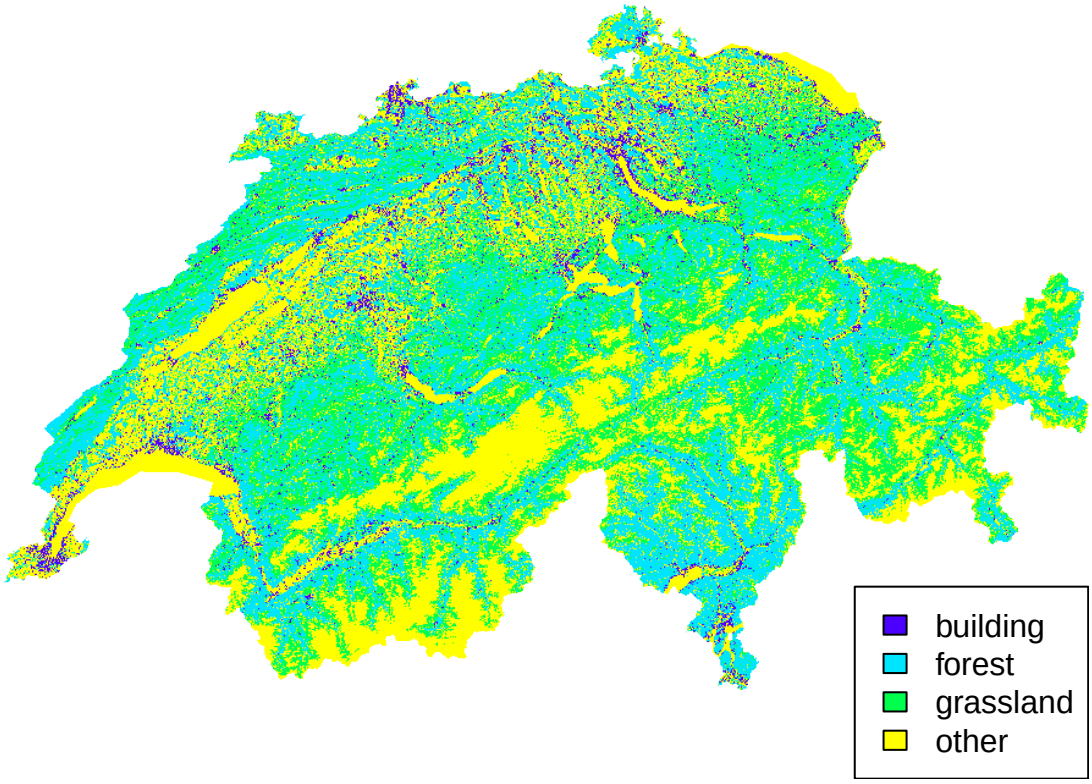

Figure 19: Map of land use in Switzerland in 2009 (raster) according to super-categories. Classification made by SFSO according to above table.

2.3.3.2 Sum of forests

First we reate forest location raster for 2009 and plot:

```
forest09 <- glu09 == 2
par(mar = c(0, 0, 0, 0))
plot(forest09, legend = F, axes = F, box = F)
```

Figure 20: Map of forest locations (raster) in Switzerland in 2009.

We are interested in quantifying the amount of forest in a given 1000m x 1000m raster cell so we will use the function `aggregate` and pass *sum* to the argument *fun* (function) so that we obtain a count of the number of 100m x 100m cells classified as forest inside each 1000m x 1000m cell.

We now aggregate forest raster to count number of 100m x 100m forest cells inside each 1000m x 1000m cell with aggregate function and align resulting sum of forests raster with elevation raster using the resample function:

```
sum.forest09 <- aggregate(forest09, fact = 10, fun = sum)
sum.forest09 <- resample(sum.forest09, dem)
values(sum.forest09)[which(values(sum.forest09) < 0)] <- 0
```

Now we plot aggregate and aligned sum of forests raster:

```
par(mar = c(0, 0, 0, 0))
plot(sum.forest09, legend = T, axes = F, box = F)
```

Figure 21: Map of sum of forests (raster). This raster counts the number of 100m x 100m cells with a forest land-use in 2009 within each 1000m x 1000m cell.

2.3.4 Landscape diversity

We will compute the landscape diversity of all 1000m x 1000m cells. Landscape diversity provides a metric of the variety of habitats (e.g. normal forest, low-canopy forest, natural grassland etc) available to birds. A more diverse 1000m x 1000m cell is expected to support a higher diversity of birds.

There are two main dimensions of diversity, richness and evenness, which we will measure with four indicators:

- **Richness**: Number of different species per unit area, and
- **Evenness**: Distribution of species, among those present.

The 2009 land use raster's original resolution is 100m x 100m so we have that 100 cells at the sharper resolution make up a single cell at the coarser resolution of 1000m x 1000m at which we are building the response variable and predictors. Let:

- n_i for $i \in \{1, 2, ..., k = 72\}$ be the number of cells with land use i, and
- $N = 100$ be the total number of cells.

Then we can calculate the different indicators according to the following table:

Indicator	Richness	Evenness	Formula
Number of species	✓	✗	$\sum_{i=1}^{k} \mathbb{1}_{\{n_i>0\}}(n_i)$
Gini	✗	✓	$\frac{\sum_{i=1}^{k}\sum_{j=1}^{k}\lvert n_i - n_j\rvert}{2\sum_{i=1}^{k}\sum_{j=1}^{k} n_i}$

Indicator	Richness	Evenness	Formula
Simpson	✓	✓	$\frac{\sum_{i=1}^{k} n_i(n_i-1)}{N(N-1)}$
Shannon	✓	✓	$-\sum_{i=1}^{k} \frac{n_i}{N} \log \frac{n_i}{N}$

Table 6: Diversity indicators

The *fun* argument of the `aggregate` function can be defined by the user. In subsection 2.3.4.1 we calculate the diversity indicators for each 1000 x 1000 cell by passing to the argument *fun* the appropriate functions.

2.3.4.1 Number of different land-uses

We first count the number of unique land-use categories (2009) of 100m x 100m cells within each 1000m x 1000m cell:

```
num.diff <- function(x, na.rm) {
    aux <- x[which(!is.na(x))]
    return(length(unique(aux)) * length(aux)/length(x))
}
div.num.diff <- aggregate(lu09, fact = 10, fun = num.diff)
```

Now we align grid of newly created raster with that of elevation grid using resample function:

```
div.num.diff <- resample(div.num.diff, dem)
pander(summary(values(div.num.diff)))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
-5.783	0	3.89	5.777	10.94	30.24	6775

We can see that the newly created and aligned raster has some negative values which is due to the interpolation of values necessary to align the raster grids. Our diversity indicators do not take values less than zero so we assign a zero value to negative valued cells.

Finally we set value of currently negative valued raster cells to zero:

```
values(div.num.diff)[which(values(div.num.diff) < 0)] <- 0
div.num.diff <- mask(div.num.diff, dem)
```

2.3.4.2 Simpson indicator (also known as Herfindahl indicator)

We calculate the Simpson indicator for each 1000m x 1000m cell using all 100m x 100m cells within (we also align raster grids and set negative values to zero):

```
simpson <- function(x, na.rm) {
    aux <- x[which(!is.na(x))]
    tab <- table(aux)/length(aux)
```

```
    n <- length(tab)
    simpson <- sum(tab^2)
    simpson <- (simpson - 1/n)/(1 - 1/n)
    return(simpson)
}
div.simpson <- aggregate(lu09, fact = 10, fun = simpson)  #align grids
div.simpson <- resample(div.simpson, dem)
# summary(values(div.simpson))
values(div.simpson)[which(values(div.simpson) < 0)] <- 0  #set negative values to zero
```

2.3.4.3 Gini indicator

We calculate the Gini indicator for each 1000m x 1000m cell using all 100m x 100m cells within (we also align raster grids and set negative values to zero):

```
gini <- function(x, na.rm) {
    aux <- x[which(!is.na(x))]
    tab <- table(aux)
    n <- length(tab)
    if (n > 0) {
        G <- sum(sapply(1:n, function(j) sapply(1:n, function(i) abs(tab[i] -
            tab[j]))))/(2 * n * sum(tab))
    } else {
        G <- NA
    }
    return(G)
}
div.gini <- aggregate(lu09, fact = 10, fun = gini)  #align grids
div.gini <- resample(div.gini, dem)
# summary(values(div.gini))
values(div.gini)[which(values(div.gini) < 0)] <- 0  #set negative values to zero
```

2.3.4.4 Shannon indicator (entropy)

We calculate the Shannon indicator for each 1000m x 1000m cell using all 100m x 100m cells within (we also align raster grids and set negative values to zero):

```
shannon <- function(x, na.rm) {
    aux <- x[which(!is.na(x))]
    tab <- table(aux)/length(aux)
    Shannon <- -sum(tab * log(tab))
    return(Shannon)
}
div.shannon <- aggregate(lu09, fact = 10, fun = shannon)
div.shannon <- resample(div.shannon, dem)  #align grids
# summary(values(div.shannon))
values(div.shannon)[which(values(div.shannon) < 0)] <- 0  #set negative values to zero
div.shannon <- mask(div.shannon, dem)
```

2.3.4.5 Compare land-use diversity indicators

First lets plot all four land-use diversity indicator rasters:

```
par(mar = c(3, 0.1, 1, 0.1), mfrow = c(2, 2))
plot(div.num.diff, axes = F, box = F, col = brewer.pal(9, "YlOrRd"),
    legend = F)
plot(div.num.diff, legend.only = T, horizontal = T, add = T,
    col = brewer.pal(9, "YlOrRd"), smallplot = c(0.25, 0.75,
        0.08, 0.1))
plot(div.simpson, axes = F, box = F, col = brewer.pal(9, "YlOrRd"),
    legend = F)
plot(div.simpson, legend.only = T, horizontal = T, add = T, col = brewer.pal(9,
    "YlOrRd"), smallplot = c(0.25, 0.75, 0.08, 0.1))
plot(div.gini, axes = F, box = F, col = brewer.pal(9, "YlOrRd"),
    legend = F)
plot(div.gini, legend.only = T, horizontal = T, add = T, col = brewer.pal(9,
    "YlOrRd"), smallplot = c(0.25, 0.75, 0.24, 0.26))
plot(div.shannon, axes = F, box = F, col = brewer.pal(9, "YlOrRd"),
    legend = F)
plot(div.shannon, legend.only = T, horizontal = T, add = T, col = brewer.pal(9,
    "YlOrRd"), smallplot = c(0.25, 0.75, 0.24, 0.26))
```

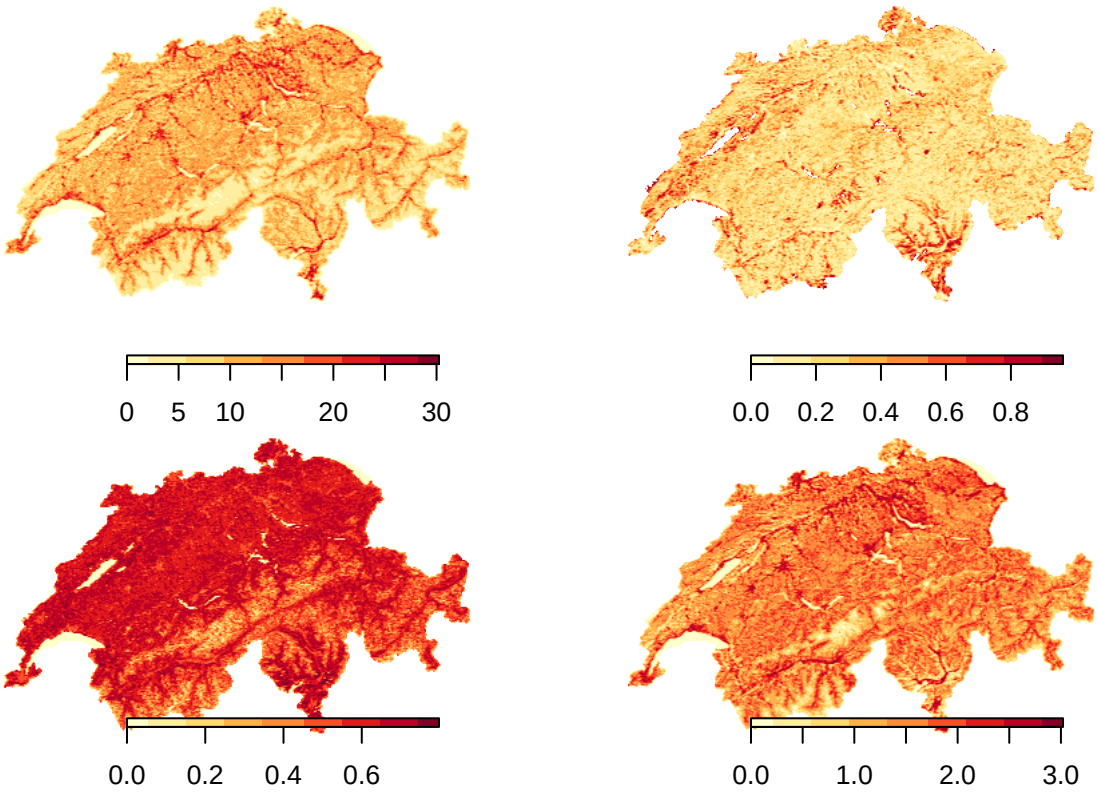

Figure 22: Map of land-use diversity (raster) in 2009 in Switzerland using four different indicators: number of different land uses (top-left), Simpson indicator (top-right), Gini indicator (bottom-left) and Shannon indicator (bottom-right). Each indicator was calculated for all 1000m x 1000m cells using all 100m x 100m cells that lie within.

We want to observe the relationship between our different diversity indicators to see if they indeed measure different dimensions of land-use diversity and we are justified in including them all in our model buiiding process.

We can use the normal function `plot` from R package `graphics` to create matrix scatter plots however the `pairs` function from the same package additionally allows us to fit a *smoothing* line to the scatter and to use the redundant *upper triangle* pannels to display *spearman* correlations.

Now we create diversity indicator dataframe with indicators as columns and locations as rows:

```
diversity.df <- cbind(as.data.frame(div.num.diff), as.data.frame(div.simpson),
    as.data.frame(div.gini), as.data.frame(div.shannon))
colnames(diversity.df) <- c("num.diff", "simpson", "gini", "shannon")
diversity.df <- na.omit(diversity.df)
# head(diversity.df)
```

Next we define function to calculate and format correlations to be displayed in upper triangle of pairs plot:

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor,
    ...) {
    usr <- par("usr")
    on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    r <- abs(cor(x, y, method = "spearman"))
    txt <- format(c(r, 0.123456789), digits = digits)[1]
    txt <- paste0(prefix, txt)
    if (missing(cex.cor))
        cex.cor <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex = cex.cor * r)
}
```

Finally we call customized matrix scatter plot:

```
pairs(diversity.df, lower.panel = panel.smooth, upper.panel = panel.cor)
```

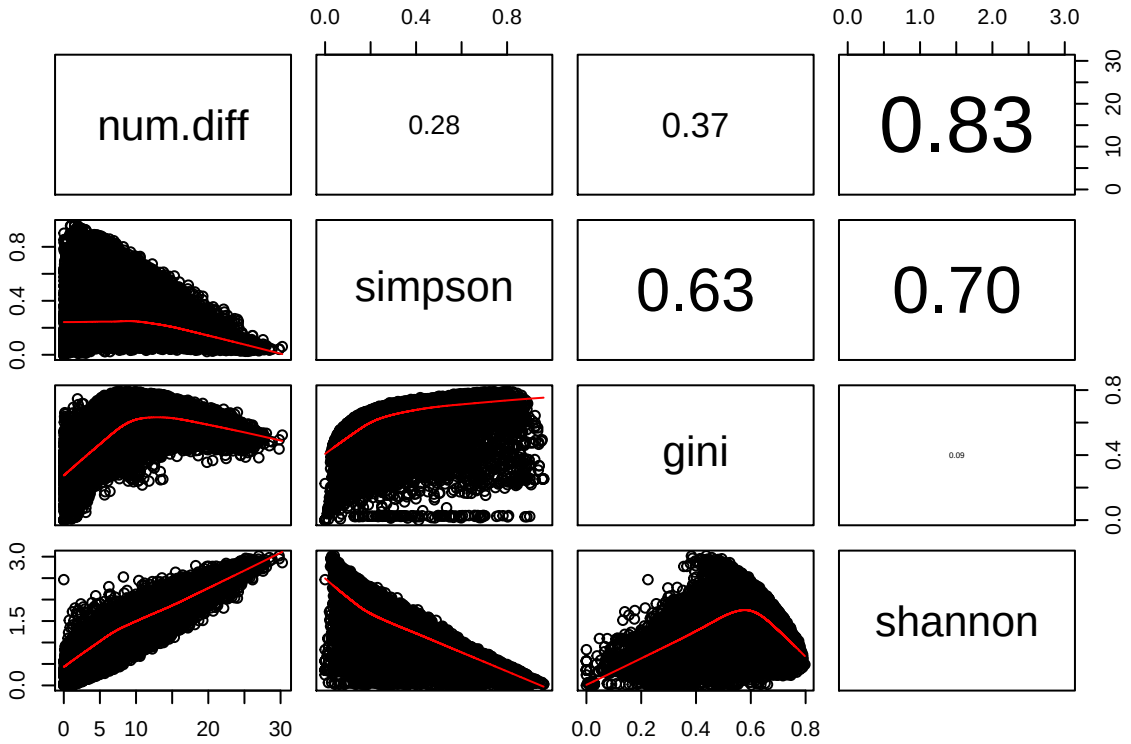

Figure 23: Scatter plots and Spearman correlations showing relationship between the four land-use diversity indicators.

```
# cor(diversity.df) cor(diversity.df, method='spearman')
```

We can see that *number of different land-uses* and the Gini indicators have a low correlation since they measure different dimensions of land-use diversity: richness and evenness respectively. Also, although the Simpson and Shannon indicators measure both evenness and richness, the Simpson indicator puts more weight on the evenness dimension, as it is more correlated to the Gini indicator than to the number of different land-uses, while the Shannon indicator puts more weight on the richness dimension, as it is more correlated to the *number of different land-uses* than to the Gini indicator. We conclude that the four indicators have sufficiently varied information to justify including them all in the generalized regression analysis of subsection 3.1.3. We could perhaps exclude either the *number of different land-uses* or the Shannon indicator as they are highly correlated and their relationship is mostly linear, however we will let the *step-wise* selection strategy of subsection 3.1.3 decide if any of the indicators should be excluded.

2.4 Water based predictors

In this section we will create a predictor of distance to water. Water is an important habitat and food source for many bird species and therefore distance to water might be an important predictor of bird species richness.

2.4.1 Rivers and Lakes

We want to create a distance to water raster however we only have available shapefiles of the lakes and rivers of Switzerland. The strategy will be to load lakes and rivers shape files, transform them into rasters, combine them into a *water* raster, and then calculate the distance to water for every cell in the 1000km x 1000km raster grid. We will later use the lake raster to identify potential reserve sites since one of the criteria will be

to avoid locating the reserve site on a lake. Both lakes and rivers shapefiles were obtained from the Swiss Federal Office of Topography (swisstopo).

The `rasterize` function from the R package `raster` can be used to make a raster based on shape object (SpatialPolygons, SpatialLines, SpatialPoints, SpatialPolygonsDataFrame, etc). It can be used by supplying the following parameters:

- **x** A shape object (SpatialPolygons, SpatialLines, SpatialPoints, SpatialPolygonsDataFrame, etc) to be converted to a raster,
- **y** A raster object, which provides the *template* raster grid for new raster,
- **field** A vector having the same length as the number of *shapes* in x. It can be one of the associated dataframe columns in case of a SpatialPolygonsDataFrame type shape object or a separate vector, and,
- **fun** A function which will act on the field argument.

For all *shape* objects in x that fall within a certain cell of y, the function will summarize the values of *field* corresponding to those shapes and assign this value to the cell. This is done for all cells. The result is a raster with the same grid as y. If function *fun* is not specified it defaults to *last* which takes the value of *field* corresponding to the last shape object in x that falls in the given cell of y.

A polygon is considered to fall within a cell if it covers the center. A line is considered to fall within a cell if any part of it goes inside the cell.

The `readOGR` function from the R package `rgdal` reads can be used to read shape files (file.shp). The data source name (*dsn* argument) is the folder (directory) where the shapefile is, and the layer is the name of the shapefile (without the .shp extension).

When certain polygon files, such as the lakes of Switzerland for example, are not available are not available it is possible to create or *draw* them oneself using a *guiding* image, such as a satellite image or a topographic map, and the `drawPoly` function from the R package `raster`. The function should be called, without arguments, after the *guiding* image has been plotted. One clicks on successive points that define the desired polygon and double-clicks when finished. We display the necessary code although, since the procedure is interactive in nature, we will not see the results.

The `writeRaster` function from the R package `raster` can be used to save a given raster object, given as first argument, in *raster* format (file.grd) by choosing *raster* value for the argument *format*. The name of the filename should be given as a string value for the argument *filename*.

First we load topographic map of Switzerland:

```
topo <- raster("../data/st_cn500/krel500.tif")
```

Next we plot topographic map then draw, for example, lac Leman and lake Zurich:

```
plot(topo)
lacLeman <- drawPoly()
plot(topo)
lakeZurich <- drawPoly()
```

We can then merge two lake polygons, convert into a lakes raster and save them:

```
lakes <- lacLeman + lakeZurich
lakes.r <- rasterize(lakes, dem, field = 1)
writeRaster(lakes.r, filename = "../data/myLakes/myLakes", format = "raster")
```

Now we plot topographic map of Switzerland and overlay cantonal borders and user-created lakes raster:

```
par(mar = c(0, 0, 0, 0))
plot(topo)
plot(ch.sp, add = T)
plot(lakes.r, add = T, col = "blue", legend = F)
```

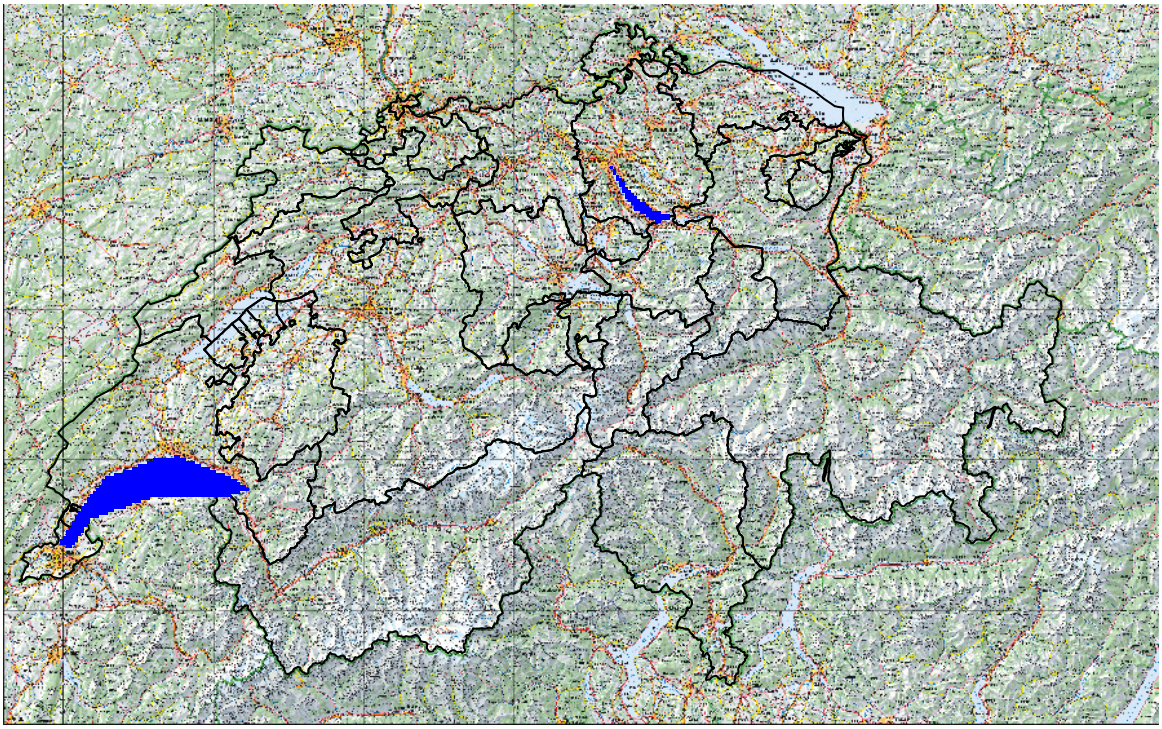

Figure 24: Topographic map (raster) of Switzerland with cantonal borders (shapefile) and Lac Leman and Lake Zurich rasters created by the user.

Next we load lakes shape file with **all** lakes of Switzerland and convert to raster using rasterize function:

```
Lakes <- readOGR(dsn = "../data/Lakes", layer = "Lakes", p4s = prj)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/Lakes", layer: "Lakes"
## with 130 features
## It has 9 fields
```

```
## Warning in readOGR(dsn = "../data/Lakes", layer = "Lakes", p4s = prj): p4s= argument given as: +init=
##  and read as: +proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333 +k_0=1 +x_0=600000 +y_0=
##  read string overridden by given p4s= argument value
```

```
Lakes.r <- rasterize(x = Lakes, y = dem, field = rep(1, length(Lakes)))
```

Now we load rivers shape file with rivers of Switzerland and convert to raster using rasterize function:

```
river <- readOGR(dsn = "../data/Rivers", layer = "Rivers", p4s = prj)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/Rivers", layer: "Rivers"
## with 125 features
## It has 6 fields
```

```
## Warning in readOGR(dsn = "../data/Rivers", layer = "Rivers", p4s = prj): p4s= argument given as: +ini
##  and read as: +proj=somerc +lat_0=46.95240555555556 +lon_0=7.439583333333333 +k_0=1 +x_0=600000 +y_0=
##  read string overridden by given p4s= argument value
```

```
river.r <- rasterize(x = river, y = dem, field = rep(1, length(river)))
```

The R package `rgeos` contains many useful functions for manipulating, transforming and exploiting shape files. Here we will use the `gUnion` function to merge the river SpatialLinesDataFrame with the Lakes SpatialPolygonsDataFrame into a *SpatialCollections object*. Subgeometries which intersect are merged into one, and those that do not are simply added to the list of shapes.

We create a water **shapefile** by joining river and lakes shape files and plot:

```
library(rgeos)  #gBoundary, gUnion, gBuffer
```

```
## rgeos version: 0.3-20, (SVN revision 535)
##  GEOS runtime version: 3.4.2-CAPI-1.8.2 r3921
##  Linking to sp version: 1.2-3
##  Polygon checking: TRUE
```

```
class(river)
```

```
## [1] "SpatialLinesDataFrame"
## attr(,"package")
## [1] "sp"
```

```
class(Lakes)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
water <- gUnion(river, Lakes)
class(water)
```

```
## [1] "SpatialCollections"
## attr(,"package")
## [1] "rgeos"
```

```
par(mar = c(0, 0, 0, 0))
plot(dem, axes = F, box = F)
plot(water, col = "light blue", add = T)
```

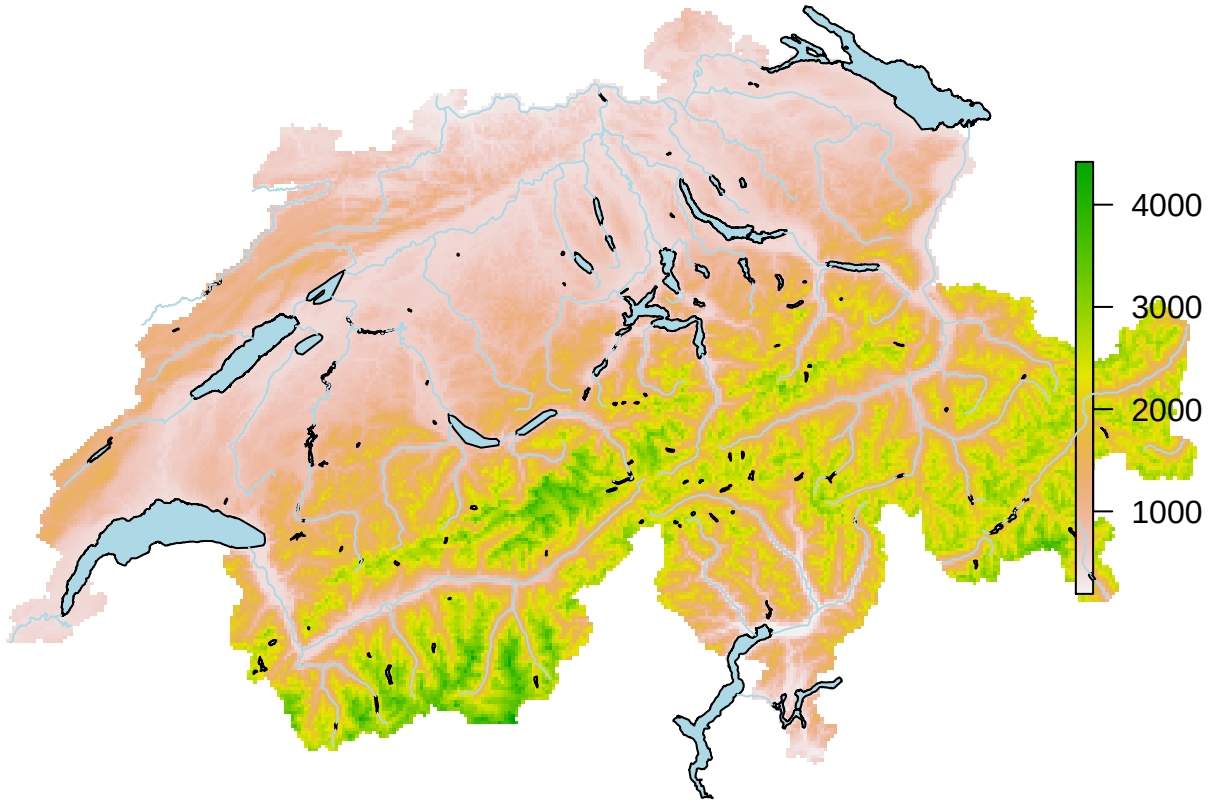

Figure 25: Map of water (lakes and rivers) in Switzerland plotted as shapes over map of elevation (raster).

We create a water **raster** file by joining river and lakes rasters and plot:

```
water.r <- river.r | Lakes.r
par(mar = c(0, 0, 0, 0))
plot(dem, axes = F, box = F)
plot(water.r, col = "light blue", add = T, legend = F)
```

Figure 26: Map of water (lakes and rivers) in Switzerland plotted as raster over map of elevation (raster).

Notice that for rasters having values of 0/1, or FALSE/TRUE, the logical operators "|" (or) can be used to create the union raster. Likewise, the logical operato "&" (and) can be used to create intersection rasters, and logical operators such as ">,<=,==" (greater than, less than or equal to, equal to) can be used to create rasters consisting of logical values.

2.4.2 Water buffer

We will create a water *buffer* shapefile showing polygons that enclose all locations that are either water locations (lakes or rivers) or are within a 2km distance of a water location. This will be done for illustrative purposes and not to build any bird species richness predictors.

To create a buffer with respect to a shape object it is preferable to work with a shape file such as a SpatialPolygon than with a raster object. Rasters are not actually shape objects although they can be used to represent them by indicating which cells are occupied by the shape. However, since rasters use a grid, the outline of the shape will always be *jagged* and the accuracy of the shape representation dependent on the resolution of the grid. On the other hand, working with shape files can be expensive if the number of shapes is large. In these examples the number of shapes is manageable so we will work with the `gBuffer` function from the `rgeos` package which works with shape objects. Further on we will create a buffer for buildings, of which there are a great quantity, and so we will work with the `buffer` function from the `raster` package which works on raster objects.

The `gBuffer` function takes two main parameters, a shape object, and the width, in the units of the coordinate reference system, in our case meters. It expands the shape object to include the area within the specified width.

First we create a shape object with a 2km buffer for water bodies in Switzerland:

```
WaterBuffer <- gBuffer(water, width = 2000)
```

Now we plot cantonal borders of Switzerland and overlay water buffer shapefile and water shapefile:

```
par(mar = c(0, 0, 0, 0))
plot(ch.sp, axes = F)
plot(WaterBuffer, col = "light blue", add = T)
plot(water, add = T, col = "blue")
```

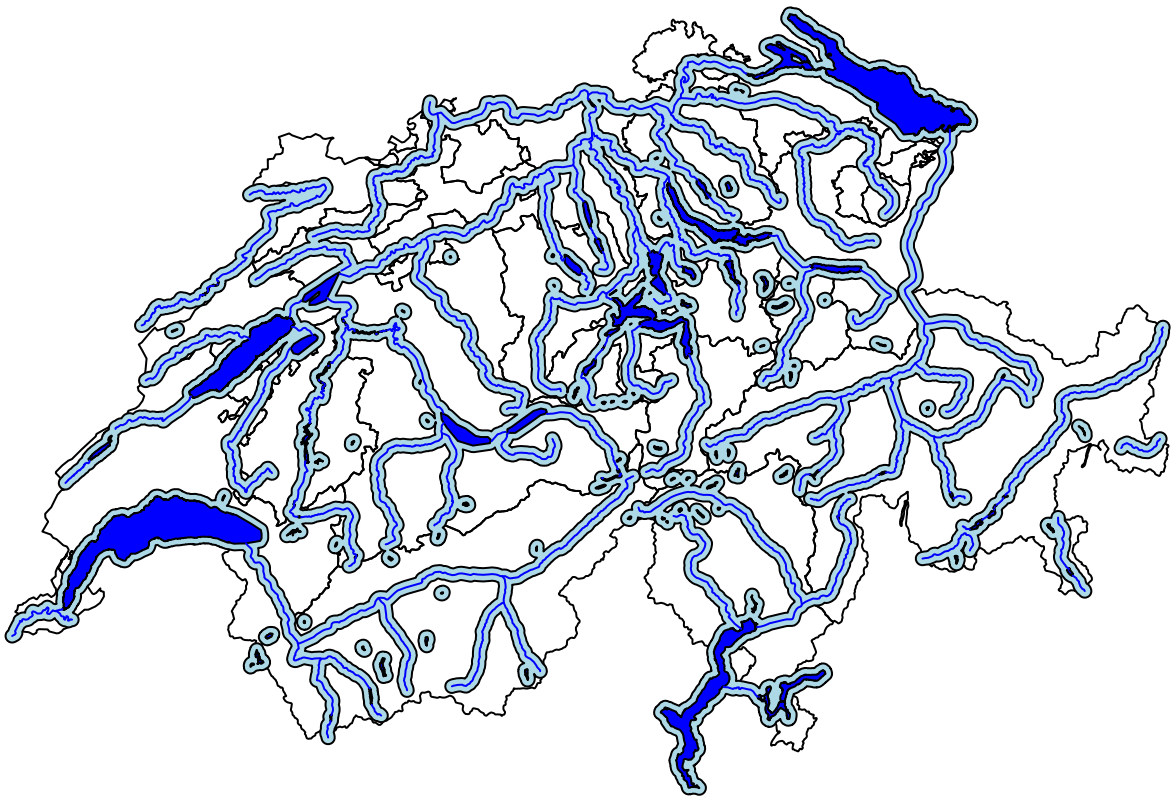

Figure 27: Map of water (lakes an rivers) in Switzerland (shapefile) and 2 km river buffer (shapefile).

Question 2.4.1 *How many bird richness observations are less than 500m distant from the rivers?*

2.4.3 Distance to water

We now build the distance to water raster to use as a predictor of bird species richness.

If a single raster is used as argument for the `distance` function, from the package `raster`, it computes the distance, for all cells that are NA, to the nearest cell that is not NA, in the units of the underlying coordinate reference system. Cells that are not NA are assigned a value (distance) of zero. Since our water raster only has TRUE values for cells where there is a river or lake we can use this function to calculate the distance from each non-water cell to the nearest water cell.

We first compute distance to water using distance function:

```
water.dist <- distance(water.r)
```

Since all cells that are not water have a value of NA in the water raster, the distance function also calculated the distance to water for cells that fall outside of Switzerland.

Now we assign NA to cells that fall outside of Switzerland by using the mask function with the elevation raster as our template:

```
water.dist <- mask(water.dist, dem)
```

Finally we plot distance to water raster:

```
par(mar = c(0, 0, 0, 0))
plot(water.dist, box = F, axes = F, col = brewer.pal(100, "Spectral"))
```

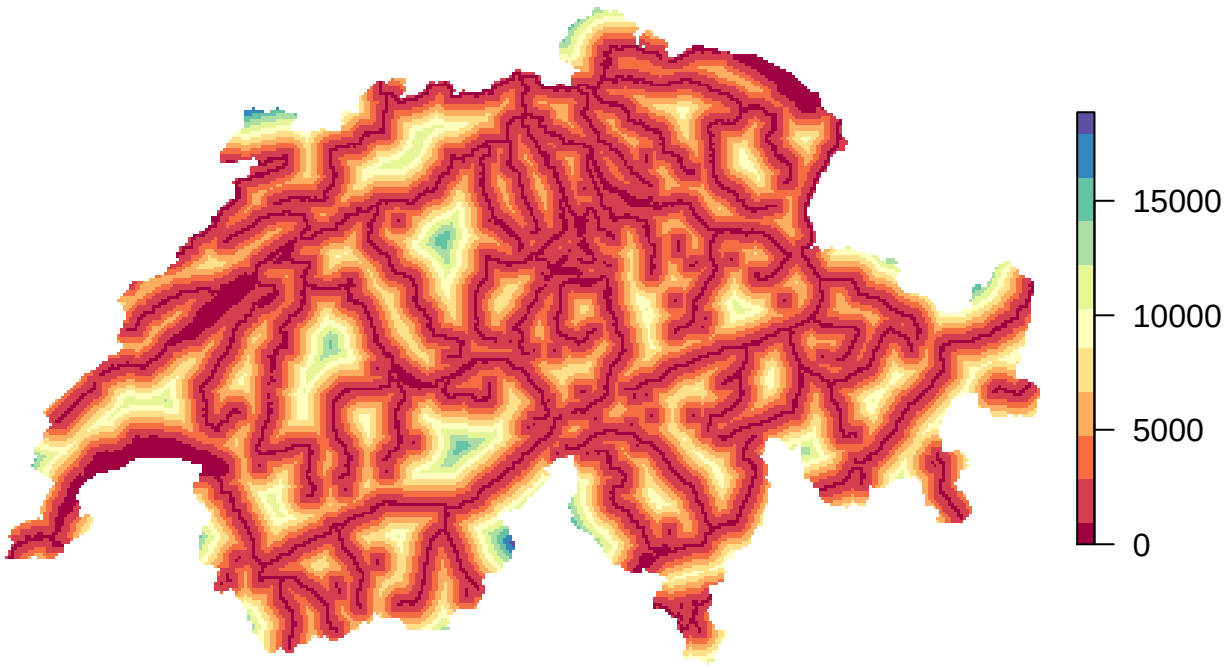

Figure 28: Map of the distance to water (lakes and rivers) plotted as a raster.

Question 2.4.2 *Observe the different layers of information we have loaded and built. What are the differences between the shape files (SpatialPolygons, SpatialLines, SpatialPoints), the raster files and dataframes? What are the resolution and origin of the different layers?*

Question 2.4.3 *How are all the predictors uploaded and created relevant in explaining bird species richness?*

3 Modeling species richness and reserve selection

3.1 Modeling bird species richness

We will buid a Genearlized Linear Model (GLM) for bird species richness based on the predictors we have loaded and built. First we review some of the theory related to GLMs as a reference for the subsequent analysis.

3.1.1 Generalized Linear Model (GLM) theory

The classic linear model can be described by the following equation:

$$y_i = \mathbb{E}[y_i|x_i] + \epsilon_i = \beta^T x_i + \epsilon_i \tag{5}$$

where:

- y is the response variable,
- x is a vector of predictors, and
- ϵ is the error term, a random variable such that:
  - mean is zero: $\mathbb{E}[\epsilon] = 0$,
  - variance is constant: $Var[\epsilon] = \sigma^2$,
  - independent identically distributed,
  - unbounded support,
  - typically we assume $y_i \sim N(\mu_i, \sigma^2)$

Although this is a useful model it has two important restrictions:

1. The range of y must not be restricted since the range of $\mathbb{E}[y|x] = \beta^T x \in (-\infty, \infty)$ is not. This is why y and ϵ are assumed to have a distribution with unbounded support such as the normal distribution.
2. The variance of y must be constant.

Here our aim is modeling bird species richness which ranges from 0 to ∞ and so, does not correspond to the properties of the classic linear regression model.

GLMs address both these issues. Like the linear model, a generalized linear model consists of a linear predictor $f(x) = \beta^T x$ and, additionally, two functions:

- A **link** function $g(\mu)$ that describes how the mean depends on the linear predictor:

  $$g(\mu) = g(\mathbb{E}[y|x]) = \beta^T x \tag{6}$$

  This allows us to have a response variable with a restricted range. The function g could, *a priori*, be any function such that if $\mathbb{E}[y|x] \in (a, b)$ then $g(\mathbb{E}[y|x]) \in (-\infty, \infty)$.
- A **variance** function $V(\mu)$ that describes how the variance depends on the mean:

  $$Var[y_i] = \phi V(\mathbb{E}[y|x]) = \phi V(\mu) \tag{7}$$

  where ϕ is a constant.

3.1.1.1 Normal response

Assume our data is distributed normally such that:

- $y_i \sim N(\mu_i, \sigma^2)$
- $\mu_i = \mathbb{E}[y_i|x_i] = \beta^T x_i$
- $Var[y_i|x_i] = \sigma^2$

Since $\mu_i \in (-\infty, \infty)$, a valid choice for the link function is:

$$g(\mu) = \mu \in (-\infty, \infty) \tag{8}$$

Also since $Var[y_i|x_i] = \sigma^2$ we have that $V(\mu) = 1$. Since a normal variable y can be written as the sum of its expectation μ and a zero mean normal with the same variance σ^2 as y we see that for this choice of link function we recover the classic linear model:

$$y_i = \mu_i + \epsilon_i = g^{-1}(\beta^T x_i) + \epsilon_i = \beta^T x_i + \epsilon_i \tag{9}$$

where $\epsilon \sim N(0, \sigma^2)$.

3.1.1.2 Bernoulli response

Assume our data is distributed Bernoulli such that:

- $y_i \sim Bernoulli(p_i)$
- $p_i = \mathbb{E}[y_i|x_i] = \beta^T x_i$
- $Var[y_i|x_i] = p_i(1 - p_i)$

Since $p_i \in (0, 1)$, a valid choice for the link function is the *log-odds ratio* or *logit* function:

$$g(p) = \log \frac{p}{1-p} \in (-\infty, \infty) \tag{10}$$

Also since $Var[y_i|x_i] = p_i(1 - p_i)$ we have that $V(p) = p(1 - p)$. For this choice of link function we obtain the **logistic** regression model:

$$\mathbb{E}[y_i|x_i] = p_i = g^{-1}(\beta^T x_i) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \tag{11}$$

$$Var[y_i|x_i] = p_i(1 - p_i) \tag{12}$$

Another valid choice for the link function is the *probit* function:

$$g(p) = \Phi^{-1}(p) \in (-\infty, \infty) \tag{13}$$

where $\Phi(p)$ is the normal cumulitive distribution function. This choice of link function leads to the **probit** regression model.

For species distribution models where the response variable indicates the presence ($y_i = 1$) or absence ($y_i = 0$) of a species at sampled location a bernoulli response variable is appropriate and, *a priori* both the *logit* or *probit* functions are valid choices for the link function.

3.1.1.3 Poisson response

Assume our data is distributed Poisson such that:

- $y_i \sim Poisson(\lambda_i)$
- $\lambda_i = \mathbb{E}[y_i|x_i] = \beta^T x_i$
- $Var[y_i|x_i] = \lambda_i$

Since $\lambda_i \in (0, \infty)$, a valid choice for the link function is the *log* function:

$$g(\lambda) = \log \lambda \in (-\infty, \infty) \tag{14}$$

Also since $Var[y_i|x_i] = \lambda_i$ we have that $V(\lambda) = \lambda$. For this choice of link function we obtain the **Poisson** regression model:

$$\mathbb{E}[y_i|x_i] = \lambda_i = g^{-1}(\beta^T x_i) = e^{\beta^T x_i} \tag{15}$$

$$Var[y_i|x_i] = \lambda_i = g^{-1}(\beta^T x_i) = e^{\beta^T x_i} \tag{16}$$

For species richness models where the response variable indicates the number of different species ($y_i = 0, 1, 2, ...$) of a species at a sampled location a poisson response variable is appropriate and, the *log* function is a valid choice for the link function.

3.1.1.4 Residual analysis and diagnostics

In the classic linear regression setting the residuals are a *natural* quantity to study owing to the fact that the response variable can be expressed as a location model:

$$y_i = \mathbb{E}[y_i|x_i] + \epsilon = \mu_i + \epsilon \tag{17}$$

meaning that the residuals $r_i = y_i - \hat{\mu}_i$ are estimators of the error term ϵ. As we have seen in subsection **??** in the linear regression setting we can look at certain plots of the residuals to verify model assumptions:

- The Tukey-Anscombe plot to check the unbiasedness of the model: $\mathbb{E}[r_i] = 0$
- The Scale-location plot to check the homoscedasticity of errors: $Var[r_i] = k$
- The normal Q-Q plot helps check that error term is distributed normally
- Cook and leverage plots help identify outliers: observations with atypical values for predictors (high leverage) and which have high influence on model estimation (high Cook's distance)..

For general linear models, in general we don't know how the residual term $r_i = y_i - \hat{\mu}_i$ is distributed since non-normal response variables y cannot, in general, be expressed as a location model $y = \mu + \epsilon$. We do however know that if the variance of the response variable is non-constant, as in the Bernoulli and Poisson case, then the residuals will also have non-constant variance. We also know that the residuals need not be distributed normally so the Scale-location and normal Q-Q plots do not apply in the GLM setting.

In order to check for the unbiasedness of the model we look at a modified Tukey-Anscombe plot constructed with *Pearson* residuals which are the residuals standardized by dividing by the estimated standard deviation of the response variable. Standardizing the residuals helps us to visualize whether the expected value of the residuals is zero for any given level of the linear predictor. The Cook and leverage plots to help identify outliers also applies for GLMs.

3.1.1.5 GLMs in R

We use the `glm` function from the `stats` package to fit our generalized linear model. Like the `lm` function the `glm` function needs at least *formula* and *data* parameters where:

- **formula** could, for example, be y~poly(x,2)+ I(z^2) + I(w==0) + I(w>0):I(x<0) + I(w>0)*I(z<0), where,
  - the I() term is used to transform existing variables,
  - the poly(x,n) term, creates a polynomial of degree n: $x + x^2 + ... + x^n$,
  - I(w>0):I(x<0) term, creates an interaction term between the indicator functions $1_{w>0}(w)$ and $1_{x<0}(x)$, and
  - I(w>0)*I(z<0) term, creates the indicator functions $1_{w>0}(w)$ and $1_{z<0}(x)$ and additionally their interaction.
- **data** indicates the data.frame where the x,y,z,w variables are located.

Additionally `glm` needs a *family* parameter which indicates the type of dependent variable and the link function to be used. For example:

- family= "binomial" or family=binomial("logit") indicates a binary dependent variable with logit link function,
- family= binomial("probit") indicates a binary dependent variable with probit link function, and
- family= poisson indicates a poisson dependent variable (values in 0,1,2,...) with a log link function.

As in the case with other R fitting functions which create *model* objects, the glm objects created by the fitting function `glm` has available certain standard functions:

- `summary`: gives a summary of estimated parameters, their significance and the overall significance of the model,
- `plot`: helps visualize he fitted model and/or gives model diagnostics to assess the validity of model assumptions, and
- `predict`: applies the model to new data points, provided that the model predictors are available, to predict the dependent variable (and probabilites in the case of glm objects).

- `coef`: provides model coefficients (the β's in the case of linear and generalized linear models).
- `fitted`: provides fitted values (the $\hat{y}$'s)
- `residuals`: provides model residuals.

3.1.2 Extraction

We will now fit a GLM model for bird species richness based on the predictors loaded and built previously.

The predictors of bird species richness considered are the following:

1. Elevation based predictors:
   a. elevation (dem)
   b. estimated slope, (slope)
2. Climate based predictors:
   a. estimated temperature (temperature.r)

b. estimated solar radiation (radiation)
c. moisture index (MIND)

3. Landscape diversity predictors:
   a. landscape diversity - no. of different land-uses (div.num.diff)
   b. landscape diversity - Simpson indicator (div.simpson)
   c. landscape diversity - Gini indicator (div.gini)
   d. landscape diversity - Shannon indicator (div.shannon)
4. Forest predictors:
   a. total forest edges (edge.sum)
   b. total forest (sum.forest09)
5. Water based predictors:
   a. distance to water (water.dist)

We create a matrix with response variable and predictors at locations where the response variable was sampled:

```
bird.model.mat <- cbind(bird, elev = extract(dem, bird[, c("X",
    "Y")]), est.slope = extract(slope, bird[, c("X", "Y")]),
    est.temp = extract(temperature.r, bird[, c("X", "Y")]), est.rad = extract(radiation,
        bird[, c("X", "Y")]), mind = extract(MIND, bird[, c("X",
        "Y")]), div.num.diff = extract(div.num.diff, bird[, c("X",
        "Y")]), div.simpson = extract(div.simpson, bird[, c("X",
        "Y")]), div.gini = extract(div.gini, bird[, c("X", "Y")]),
    div.shannon = extract(div.shannon, bird[, c("X", "Y")]),
    edge.sum = extract(edge.sum, bird[, c("X", "Y")]), forest.sum = extract(sum.forest09,
        bird[, c("X", "Y")]), water.dist = extract(water.dist,
        bird[, c("X", "Y")]))
```

3.1.3 Model fit

Since bird species richness is a count of the number of observed species at various locations we use a Poisson regression model.

We will make predictor-response curves, based on univariate Poisson models for bird species richness, for each of the predictors, to aid our understanding and interpretation of the relationship between bird species richness and the different predictors.

First we create a function to calculate the values of the predictor for which we wish to plot the response:

```
func1 <- function(x) {
    xi <- seq(min(as.numeric(x), na.rm = T), max(as.numeric(x),
        na.rm = T), length.out = 100)
    return(xi)
}
```

Next we get equally spaced representative values of all predictors:

```
toplot <- apply(bird.model.mat, 2, func1)
colnames(toplot) <- colnames(bird.model.mat)
```

Now create a vector of strings with all predictor names to aid in building the GLM formula:

```
predictors <- c("elev", "est.slope", "est.temp", "est.rad", "mind",
    "div.num.diff", "div.simpson", "div.gini", "div.shannon",
    "edge.sum", "forest.sum", "water.dist")
```

Finally we fit a univariate Poisson regression model for each predictor and use this model to get an estimate of bird richness for the values of the predictors calculated previously and plot response curves:

```
par(mar = c(3, 3, 3, 3), mfrow = c(4, 4))
for (var in predictors) {
    # choose predictor
    vari <- bird.model.mat[, var]
    mi <- glm(as.formula(paste("richness ~ poly(", var, ",2)")),
        family = poisson, data = na.omit(bird.model.mat[, c("richness",
            var)]))
    plot(vari, bird.model.mat$richness, ylab = "richness", xlab = "",
        main = var)
    toploti <- data.frame(toplot[, var])
    colnames(toploti) <- var
    lines(y = predict(mi, newdata = toploti, type = "response"),
        x = toplot[, var], col = "red")
}
```

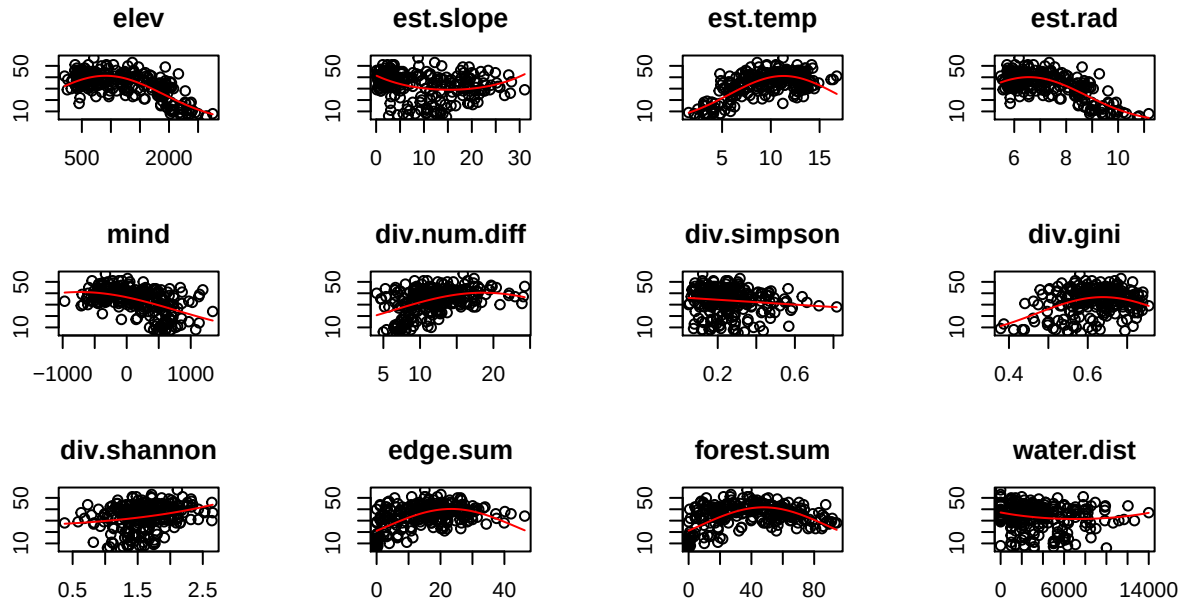

Figure 29: Univariate predictor vs. response scatter plots and fitted model curves. Red lines indicate the number of bird species for a given level of the independent variable as predicted by corresponding univariate GLM.

Notice that the univariate predictor-response curves imply that, the higher the temperature the more bird richness there is, at least up until a certain threshold temperature after which the bird richness decreases again.

Now we fit a Poisson regression model with all the predictors that are available:

```
form <- as.formula(paste("richness", paste(paste("poly(", predictors,
    ",2)", sep = ""), collapse = " + "), sep = " ~ "))
m1.glm <- glm(form, data = na.omit(bird.model.mat), family = poisson,
    maxit = 100)
```

Including too many predictors in the model can lead to overfitting the data and poor generalization of the model. On the other hand including too few predictors can lead to sytematic errors and bias. This is the bias-variance trade-off inherent to modeling in limited data settings. We try and strike a balance by minimizing the Akakike Information Criterion (AIC) using the backward-forward stepwise selection strategy. The AIC has two components:

- **log-likelihood**: higher log-likelihoods are rewarded with a lower AIC score, while
- **number of predictors**: higher number of predictors are penalized with a higher AIC score.

The backward-forward stepwise selection strategy consists of starting with all predictors and at each step calculating the resulting AIC of:

1. Omitting any of the predictors currently included in the model,
2. Adding any of the predictors currently excluded from the model, and,
3. Leaving the model as it is.

If the lowest AIC results from taking action from 1 or 2 the corresponding predictor is omitted or added to the model and we perform the next step. The stepwise search ends when the lowest AIC results from point 3: leaving the model as it is.

In R backward-forward stepwise selection is implemented in the `step` function by choosing the value *both* for the *direction* parameter.

We perform backward and forward stepwise selection of variables using step function:

```
m1.glm.step <- step(m1.glm, trace = F, direction = "both")
pander(m1.glm.step$anova, caption = "Summary of backward-forward stepwise selection")
```

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	235	268.8	1691
- poly(div.simpson, 2)	2	0.04473	237	268.9	1687
- poly(water.dist, 2)	2	0.2433	239	269.1	1684
- poly(elev, 2)	2	1.345	241	270.5	1681
- poly(est.slope, 2)	2	1.805	243	272.3	1679

Table 8: Summary of backward-forward stepwise selection

```
pander(summary(m1.glm.step)$coefficients, caption = "Summary of step glm model fit.")
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.457	0.01152	300	0
poly(est.temp, 2)1	-0.1183	0.8488	-0.1393	0.8892
poly(est.temp, 2)2	-0.6839	0.3664	-1.866	0.06199
poly(est.rad, 2)1	-3.241	0.8263	-3.923	8.761e-05
poly(est.rad, 2)2	-1.625	0.4521	-3.594	0.0003261
poly(mind, 2)1	-0.8744	0.2571	-3.401	0.0006704
poly(mind, 2)2	-0.1966	0.1882	-1.045	0.2962
poly(div.num.diff, 2)1	-0.7194	0.8835	-0.8143	0.4155
poly(div.num.diff, 2)2	-0.5101	0.3288	-1.551	0.1208
poly(div.gini, 2)1	1.082	0.6772	1.598	0.11
poly(div.gini, 2)2	-0.4931	0.2705	-1.823	0.06835
poly(div.shannon, 2)1	1.707	1.142	1.494	0.1351
poly(div.shannon, 2)2	0.972	0.3119	3.117	0.00183
poly(edge.sum, 2)1	0.3103	0.3466	0.8952	0.3707
poly(edge.sum, 2)2	-0.679	0.2266	-2.997	0.002728
poly(forest.sum, 2)1	0.2555	0.3515	0.727	0.4672
poly(forest.sum, 2)2	-1.401	0.3047	-4.599	4.244e-06

Table 9: Summary of step glm model fit.

Three out of the four diversity indices were selected covering both evenness and richness dimensions of diversity suggesting both are important for bird species richness. All coefficients related to weather variables are negative meaning the model predicts greater bird species richness in cooler, dryier and shadier locations. This despite the fact that in the univariate analysis it seemed as though temperature had a positive effect on bird species richness, at least up until a certain threshold temperature after which the bird richness decreases again. However, univariate analysis is limited since we do not adjust for the effect of other variables.

Before going further we also fit the *trivial* model with no predictors and only a constant term as the linear predictor:

$$g(\mathbb{E}[y|x]) = log(\mu) = \beta_0 \tag{18}$$

The trivial model will serve as a benchmark to check that our model achieves some improvement over simply estimating the bird species richness as the sample mean species richness regardless of location.

We fit trivial model with no predictors

```
m0.glm <- glm(richness ~ 1, data = na.omit(bird.model.mat), family = poisson("log"))
pander(summary(m0.glm)$coefficients, caption = "Summary of trivial model fit.")
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.518	0.01068	329.3	0

	Estimate	Std. Error	z value	Pr(>|z|)

Table 10: Summary of trivial model fit.

We will refer to the model with environmental predictors as the *environmental* model.

3.1.4 Model diagnostics

We want to check that our model is unbiased by creating a Tukey-Anscombe plot and checking that the Pearson residuals do not have structure when compared to the linear predictor: the level of the Pearson residuals should be more or less the same regardless of the level of the linear predictor. If we see some *pattern* in the Pearson residuals when plotted against the linear predictor it means our model is biased. We also want to create a standardized leverage vs. Cook's distance plot to identify outliers in the dataset, points which possibly don't belong to the dataset and which have a big impact on the model estimation.

The R package `boot` has some useful functions for GLM diagnostics including `glm.diag` which, among other things, calculates the Cook's statistic and leverage of each observation which we use to identify outliers. High Cook's distance indicate influential data points - data points which are influencing the value of estimations - while high leverage indicate data points with unusual values for the predictor variables. We are interested in finding and possibly investigating further those points that have unusual predictor values and are influential since these could be data points that don't belong in our data set and are affecting model estimation.

For Cook's statistic we use the threshold value of

$$\frac{8}{n-2p} \tag{19}$$

to define points with high Cook's statistic (influential points), while for leverage we us the threshold value of

$$\frac{2p}{n-2p} \tag{20}$$

to define points with high leverage (unusual x values). These threshold values are recommended in the documentation of the `glm.diag` function.

First we obtain Cook's distance and leverage for all observations using glm.diag function:

```
library(boot)  #glm.diag
# glm diagnostics- in particular for cook and leverage plot
diags <- glm.diag(m1.glm.step)
cook <- diags$cook
levg <- diags$h
stdr.levg <- levg/(1 - levg)
```

Next we set cook and leverage thresholds:

```
n <- nrow(na.omit(bird.model.mat))
p <- m1.glm.step$df.null - m1.glm.step$df.residual + 1
cook.thrsh <- 8/(n - 2 * p)
levg.thrsh <- 2 * p/(n - 2 * p)
```

Next we obtain linear predictors $\beta^T x_i$ and Pearson residuals r_i for all observations i:

```
xx.fit <- predict(m1.glm.step, type = "response")
yy.fit <- residuals(m1.glm.step, type = "pearson")
```

We now create a smoother for the relationship Pearson residuals vs. linear predictor. This will serve as an estimation of the expected Pearson residual conditional on different values of the linear predictor. If the model has no bias it should be *reasonably* constant. This will be red line of Tukey Anscombe plot and should be *reasonably* flat.

Now we create smoother of linear predictor vs. Pearson residuals:

```
ls.fit <- loess.smooth(xx.fit, yy.fit, family = "gaussian")
```

Random fluctuations of the data can cause the smoother to not be totally flat. To obtain an indication of what constitutes *reasonably* flat smoothers we destroy any structure the Pearson residuals have with respect to the linear predictor by randomly ordering them. We do this 100 times to create 100 orderings of the Pearson residuals and for each ordering create a smoother with respect to the ordered linear predictor. If our ordered Pearson residuals indeed have no structure then the smoother created above (plotted in red in the Tukey-Anscombe plot) should be similar to the 100 ceated below (plotted in grey n the Tukey Anscombe plot).

Finally we create Tukey-Anscombe and standardized levarage vs. Cookś distance plots:

```
# Tukey-Anscombe plot with pearson residuals
par(mfrow = c(1, 2))
# plot linear predictor vs. pearson residuals
plot(xx.fit, yy.fit, xlab = "linear predictor", ylab = "Pearson residuals")
for (i in 1:100) {
    # Obtain sample of Pearson residuals to destroy any structure
    # they may have.
    yy.smpl <- sample(yy.fit, replace = TRUE)
    # Create a smoother of unstructured Pearson residuals
    # vs.linear predictor. These will be flat up to random
    # variation and so represents acceptable deviations from
    # flatness
    ls.smpl <- loess.smooth(xx.fit, yy.smpl)
    # Add smoother to plot in grey
    lines(ls.smpl, col = "grey")
}
# Add smoother for real Pearson residuals (possibly having
# structure) vs. linear predictor
lines(ls.fit$x, ls.fit$y, col = "red")
abline(h = 0, lty = 3)

# Cook distance and leverage plot
plot(stdr.levg, cook, xlab = "Standardized leverage", ylab = "Cook statistic")
abline(h = cook.thrsh, v = levg.thrsh, lty = 2)
```

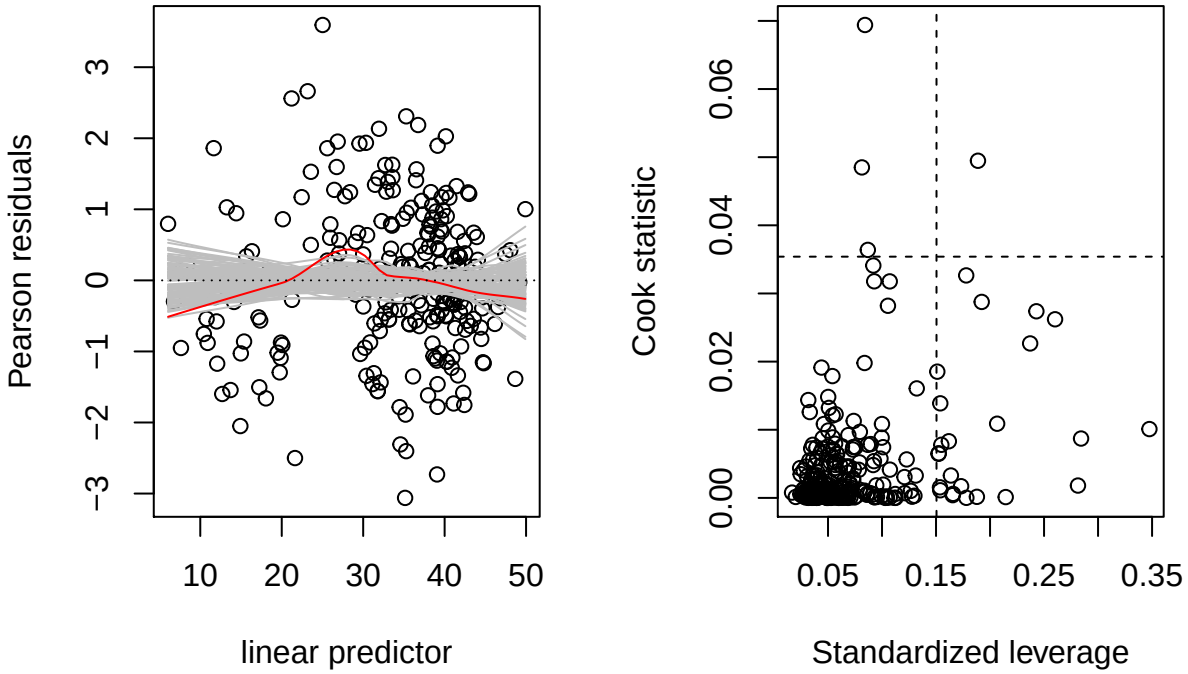

Figure 30: Generalized linear model diagnostics: Tukey-Anscombe plot (left) for identifying model bias and Cook distance vs. leverage plot (right) for identifying outliers.

We can see that the behavior of the residuals is mostly within the range of acceptable deviation from the flat mean line. There is one observation that is influential (i.e. has high Cook's distance) and has unusual values for the predictors (i.e. has high leverage).

We first identify outliers:

```
# identify influential outliers
(indx.out <- which(stdr.levg > levg.thrsh & cook > cook.thrsh))
```

```
## 191
## 190
```

```
outs <- na.omit(bird.model.mat)[indx.out, ]
```

Now we plot elevation raster and overlay bird richness observation and outlier locations:

```
# locate outliers on map
par(mar = c(0, 0, 0, 0))
plot(dem, col = terrain.colors(100), legend = F, axes = F, box = F)
points(bird$X, bird$Y, cex = bird$richness/50, pch = 21, bg = "yellow")
points(outs$X, outs$Y, cex = 3, col = "blue")
```

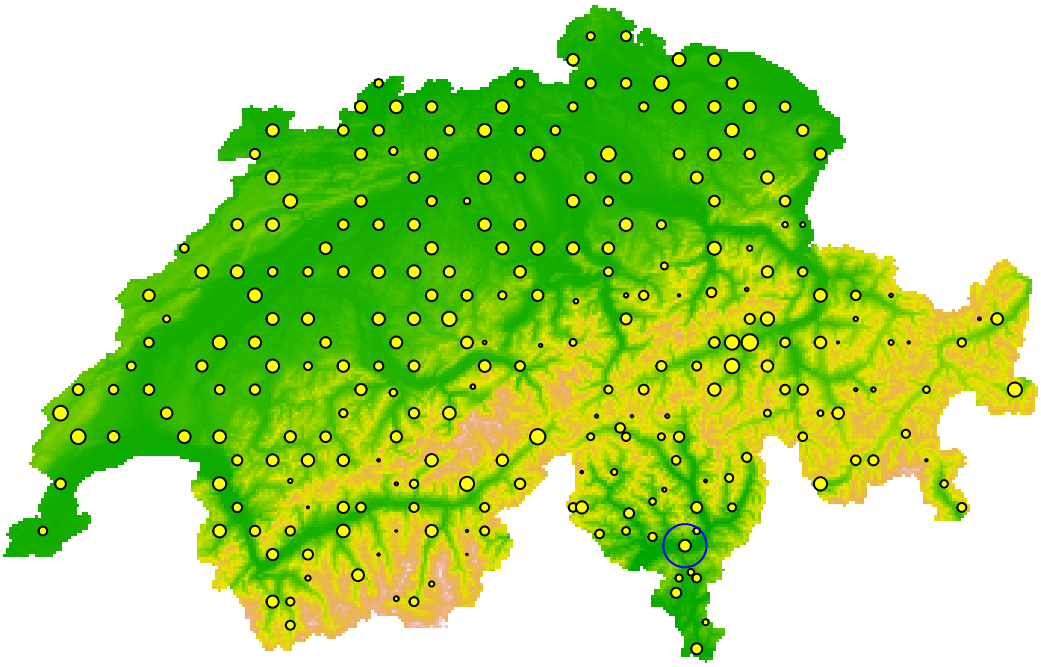

Figure 31: Observation and outlier locations plotted over map of elevation (raster). Aid in identifying if outlier locations have special characteristics not taken into account by the model.

The outlier is located in the canton of Ticino. To investigate more we could make raster plots of all the selected and non-selected predictors available to try and understand why this is a special location.

3.1.5 Model evaluation

To evaluate the accuracy of our model we will use the square root of the mean square error:

$$MSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{\mu}_i)^2}{N}} \tag{21}$$

where:

- N is the number of observations,
- y_i is the bird richness of observation i, and,
- $\hat{\mu_i}$ is the predicted bird richness of observation i by our model.

We will also evaluate the model using the Spearman correlation between the predicted bird richness and the real bird richness.

Measures of accuracy can be calculated *in-sample* or *out-of-sample*:

-**in-sample**: Model is estimated and evaluated using the same set of observations. -**out-of-sample**: Model is estimated and evaluated using different, mutually exclusive, sets of observations.

We first calculate the *in-sample* mean square error for the trivial and environmental models. We also calculate the *in-sample* Spearman correlation between fitted bird richness an real bird richness:

```
mse.in0 <- sum((m0.glm$fitted - na.omit(bird.model.mat)$richness)^2)/length(m0.glm$fitted)
mse.in <- sum((m1.glm.step$fitted - na.omit(bird.model.mat)$richness)^2)/length(m1.glm.step$fitted)
spCor.in <- cor(m1.glm.step$fitted, na.omit(bird.model.mat)$richness,
    method = "spearman")
```

In-sample measures of accuracy give us an over-optimistic estimate of error since they are calculated for the same data that was used to fit the model. We will obtain a more accurate estimate of the error by applying the modeling procedure to a sample of the data and evaluating the accuracy of the model on the remaining *hold-out* sample. We will use a random sample of 70% of the data to fit the model and the remaining 30% to evaluate its accuracy. We repeat this for 100 different random data splits to obtain a distribution of the square root of the mean square error.

Now we perform out-of-sample evaluation of model accuracy:

```
reps <- 100  #number of repetitions
ratio <- 0.7  #fraction of data to be used for model fitting
CorEval <- c()  #holder spearman correlation between predicted and real bird richness
mse.out <- c()  #holder for out of sample repetitions of MSE of environmental model
mse.out0 <- c()  #holder for out of sample repetitions of MSE of trivial model
for (i in 1:reps) {
    # create row index variable
    Sequence <- seq(from = 1, to = dim(bird.model.mat)[1])
    # Sample ratio*100 % of row indices and order
    SampCali <- sort(sample(Sequence, size = round(ratio * dim(bird.model.mat)[1]),
        replace = FALSE, prob = NULL))
    # Create matrix with response and predictors for model
    # fitting
    Birds_CALI <- bird.model.mat[c(SampCali), ]
    # Create matrix with response and predictors for model
    # evaluation
    Birds_EVAL <- bird.model.mat[-c(SampCali), ]
    # Fit environmental model with sample of 70% of data with all
    # predictors
    m1.glm_eval <- glm(form, data = na.omit(Birds_CALI), family = poisson,
        maxit = 100)
    # Perform stepwise selection on environmental model fitted
    # with 70% of data
    m1.glm.step_eval <- step(m1.glm, trace = F, direction = "both")
    # Fit trivial model with sample of 70% of data
    m0.glm_eval <- glm(richness ~ 1, data = na.omit(Birds_CALI),
        family = poisson("log"))
    # Predict bird richness for 30% of data NOT used to fit model
    # using environmental model
    pred.eval <- predict(m1.glm.step_eval, newdata = Birds_EVAL,
        type = "response")
    # Predict bird richness for 30% of data NOT used to fit model
    # using trivial model
    pred.eval0 <- predict(m0.glm_eval, newdata = Birds_EVAL,
        type = "response")
    # Calculate Spearman correlation between prediction of
    # environmental model and true bird richness
    cor.eval.glm <- cor(pred.eval, Birds_EVAL$richness, method = "spearman")
    # Store latest repetition of correlation
    CorEval <- c(CorEval, cor.eval.glm)
```

```
    # Calculate MSE of environmental model on hold-out sample
    mse.aux <- sum((pred.eval - Birds_EVAL$richness)^2)/length(pred.eval)
    # Calculate MSE of trivial model on hold-out sample
    mse.aux0 <- sum((pred.eval0 - Birds_EVAL$richness)^2)/length(pred.eval0)
    # Store leates repetition of MSE for environmental and
    # trivial models
    mse.out <- c(mse.out, mse.aux)
    mse.out0 <- c(mse.out0, mse.aux0)
}
```

Next we print in-sample correlation and a summary of 100 out-of-sample correlations:

```
spCor.in
```

```
## [1] 0.7508188
```

```
summary(CorEval)  # ~ 0.74 (mean spearman correlation)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.5996  0.6978  0.7436  0.7415  0.7801  0.8728
```

The *out-of-sample* Spearman correlation between predicted and real bird richness is reasonalby high and very similar to the *in-sample* version indicating good performance of the model and good control of overfitting.

Finally we print in-sample square root of MSE and a summary of 100 out-of-sample square root of MSEs for trivial and environmental models:

```
mse.in0^0.5
```

```
## [1] 10.98184
```

```
mse.in^0.5
```

```
## [1] 5.711558
```

```
pander(summary(mse.out0^0.5), caption = "Summary of out-of-sample square root of mean square error")
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
9.321	10.44	10.98	11.01	11.62	12.58

Table 11: Summary of out-of-sample square root of mean square error

```
quantile(mse.out0^0.5, c(0.05, 0.5, 0.95))
```

```
##        5%       50%       95%
##  9.759516 10.981693 12.312348
```

```
pander(summary(mse.out^0.5), caption = "Summary of out-of-sample square root of mean square error")
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
4.555	5.373	5.653	5.67	5.991	6.674

Table 12: Summary of out-of-sample square root of mean square error

```
quantile(mse.out^0.5, c(0.05, 0.5, 0.95))
```

```
##       5%      50%      95%
## 4.959031 5.653278 6.403525
```

According to out-of-sample estiamation the square root of the mean square error is between 4.96 and 6.4 with 90% confidence. In this case the out-of-sample estimation is similar to the usually over-optimistic in-sample estimate. This is probably because our stepwise selection process prevented our model from overfitting the data. We can also see that the environmental represents a considerable improvement on the trivial model.

Question 3.1.1 *Is your model able to predict the bird species richness in Switzerland? What could we do to improve it?*

3.1.6 Projection of model to Switzerland

We will project our model for the entire Switzerland raster grid to visualize the bird species richness score we have constructed.

First we create a matrix with predictors at all locations of the Switzerland raster grid:

```
bird.extent.mat <- cbind(as.data.frame(coordinates(dem)), elev = values(dem),
    est.slope = extract(slope, coordinates(dem)), est.temp = extract(temperature.r,
        coordinates(dem)), est.rad = extract(radiation, coordinates(dem)),
    mind = extract(MIND, coordinates(dem)), div.num.diff = extract(div.num.diff,
        coordinates(dem)), div.simpson = extract(div.simpson,
        coordinates(dem)), div.gini = extract(div.gini, coordinates(dem)),
    div.shannon = extract(div.shannon, coordinates(dem)), edge.sum = extract(edge.sum,
        coordinates(dem)), forest.sum = extract(sum.forest09,
        coordinates(dem)), water.dist = extract(water.dist, coordinates(dem)))
```

We will now project model over all Switzerland by evaluating linear predictor with observations of the matrix we have just created. We then create raster of predicted bird species richness using model prediction for each cell and its coordinates.

The function `rasterFromXYZ` can be used to create a raster from a matrix containing the x and y coordinates in the first two columns, and the corresponding raster value in the third. The x and y coordinates must be on a regular raster grid in order to obtain a raster with the desired resolution. This is because if the resolution is not specified explicitly, it is assumed to be the minimum distance between x and y coordinates. A resolution of up to 10 times smaller than the mininimum may be attempted if a regular grid can otherwise not be created.

We now use fitted model to predict bird species richness for Switzerland raster grid and create bird richness raster:

```
proj.glm <- round(predict(m1.glm.step, newdata = bird.extent.mat,
    type = "response"))
BirdRich <- rasterFromXYZ(cbind(bird.extent.mat[, c("x", "y")],
    proj.glm))
```

Finally we plot projected bird species richness raster:

```
par(mar = c(0, 0, 0, 0))
plot(BirdRich, box = F, axes = F, col = brewer.pal(9, "PuRd"))
```

Figure 32: Map of projected bird richness (raster) over Switzerland. Displays the bird species richness as predicted by the fitted GLM model.

3.2 Reserve selection

We will now identify potential bird reserve locations. The new reserve location should satisfy:

1. be a minimum of 1km from any building,
2. not overlap with existing reserves (parks.r),
3. lakes should be excluded, (lakes.r),
4. have *high* bird diversity
5. have a total area of at least 40 km^2,

3.2.1 Buildings

We will now load a shapefile of builidngs. The buildings shapefile was obtained from the Swiss Federal Office of Topography (swisstopo). We will compute a raster of number of buildings per cell. This is done as an exercise to further illustrate the use of the function `rasterize`. We also build a *buildings buffer* raster which

indicates if any given cell has buildings in it or at a distance of 1000m or less. The purpose of creating this raster is to identify the most pristine locations accross Switzerland. We will use the buildings buffer raster to identify locations which satisfy condition 1 above.

First we load buildings shape file:

```
Buildings <- readOGR(dsn = "../data/VEC200_Building", layer = "VEC200_Building",
    p4s = prj)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/VEC200_Building", layer: "VEC200_Building"
## with 313670 features
## It has 6 fields
```

```
class(Buildings)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

We will now convert buildings shapefile into a raster file where the cell value corresponds to the number of buildings *inside* each cell.

As we will see, buildings are often much smaller than the cell size meaning that the whole building may be inside a cell but, if we treat it like a polygon when we rasterize, it may not count as falling within the cell since it does not cover the cell center (see subsection 2.4.1). For this reason, we will convert polygon buildings to line buildings before rasterizing so that a building is considered to fall within a cell if any part of the line that conforms it falls inside the cell.

Now for the small extent created before (in subsection 2.3.1) we obtain the x and y coordinates of the grid lines.

```
cells_centers <- coordinates(crop(dem, extSmall))
lines_x <- unique(c(cells_centers[, 1] - 500, cells_centers[,
    1] + 500))
lines_y <- unique(c(cells_centers[, 2] - 500, cells_centers[,
    2] + 500))
```

Now we isolate the building shapes that fall into this extent.

```
fewBuildings <- crop(Buildings, extSmall)
```

Next we convert to raster. We also assign to each cell of the grid the number of building shapes that *fall* inside:

```
fewBuildings.r <- rasterize(x = as(fewBuildings, "SpatialLines"),
    y = crop(dem, extSmall), field = rep(1, length(fewBuildings)),
    fun = function(x, ...) sum(x, ...), na.rm = T)
```

Now we ssign zero values to NA valued cells since these correspond to cells with no building shapes. Cells outside Switzerland should still be NA-valued (we use mask function as usual):

```
fewBuildings.r[is.na(fewBuildings.r)] <- 0
fewBuildings.r <- mask(fewBuildings.r, crop(dem, extSmall))
# summary(values(fewBuildings.r))
```

Finally we plot grid, raster values, and building shapes.

```
plot(crop(dem, extSmall), col = "white", legend = F)
plot(fewBuildings, add = T)
abline(v = lines_x, h = lines_y, col = "red")
text(cells_centers, labels = values(fewBuildings.r), cex = 0.7,
    col = "blue")
```

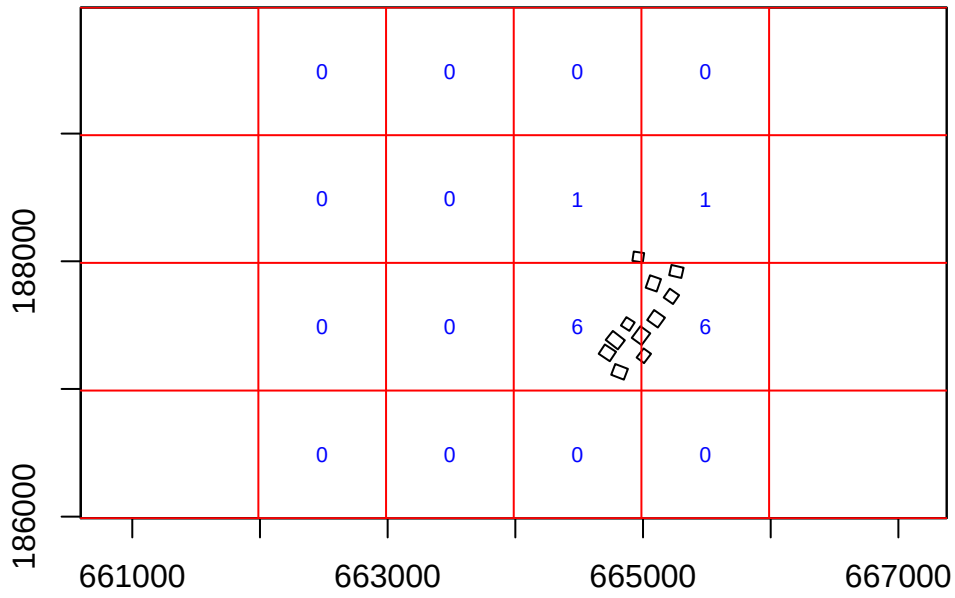

Figure 33: Map of buildings (shapefile) with sum of buildings raster values and grid displayed. Zoom in for 5000m x 5000m extent at center of entire raster.

As we can see we obtain the desired result. Each raster cell-value counts the number of buildings that touch the corresponding cell. Note that if we had used buildings as polygons shape we would obtain a different result.

Now apply above procedure to the whole Buildings SpatialPolygonsDataFrame. Since it has 313,670 polygons, this takes quite a while which is why we save result so that we may simply upload result in future.

```
BuildingsLines <- as(Buildings, "SpatialLines")
sum.buildings <- rasterize(x = BuildingsLines, y = dem, field = rep(1,
    length(Buildings)), fun = function(x, ...) sum(x, ...), na.rm = T)
projection(sum.buildings) <- prj
sum.buildings[is.na(sum.buildings)] <- 0
sum.buildings <- mask(sum.buildings, dem)
writeRaster(sum.buildings, filename = "../data/sumBuildings/sumBuildings",
    format = "raster")
```

Now we plot the resulting sum of buildings raster.

```
# summary(values(sum.buildings))
par(mar = c(0, 0, 0, 0))
plot(sum.buildings, legend = T, box = F, axes = F, col = brewer.pal(20,
    "Blues"))
```

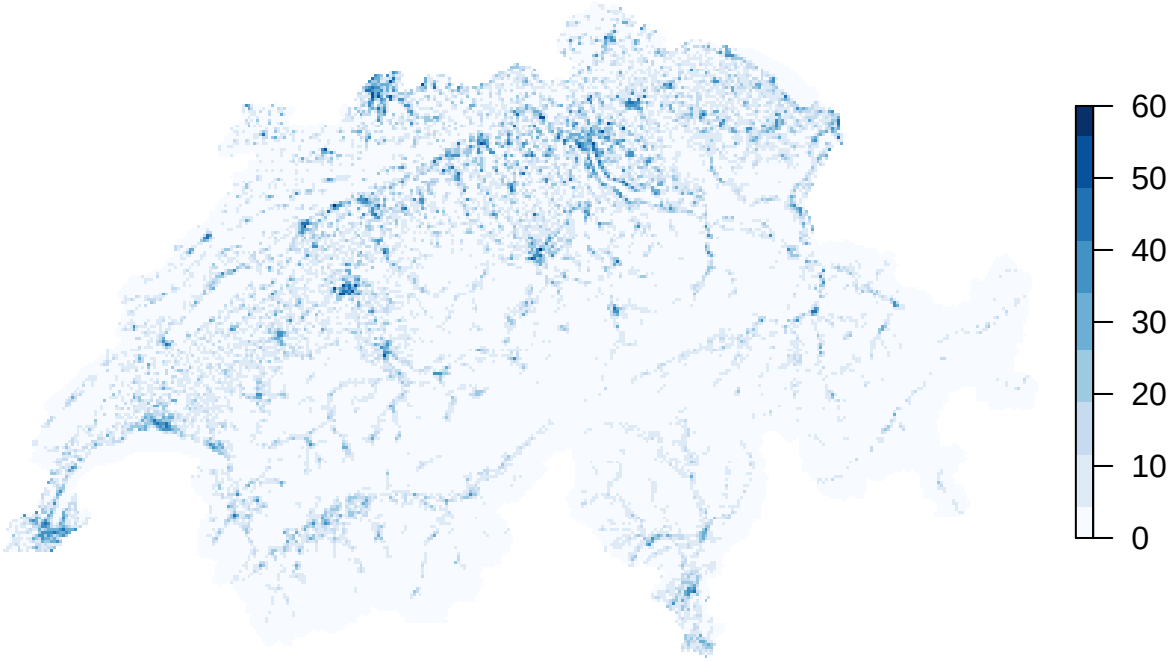

Figure 34: Map of the number of buildings (raster) in Switzerland. Count of the number buildings that fall into each 1000m x 1000m cell.

To get a close up of the location with most buildings we will again create small extent (as in subsection 2.3.1) except this time we will do it with respect to the center of the cell with most buildings. We use the function `xyFromCell` from the R package `raster` which provides the x and y coordinates of the cell numbers passed to the argument *cell*, from the raster object passed as first argument.

We will now zoom in to cells surrounding the cell with most buildings.

First we obtain the centre of cell with most buildings:

```
center <- xyFromCell(sum.buildings, cell = which(values(sum.buildings) ==
    max(values(sum.buildings), na.rm = T)))
```

Now we obtain the extent of 5 x 5 cell subset with the above center:

```
extSmall <- extent(c(center[1] - 500 - 2 * 1000, center[1] +
    500 + 2 * 1000, center[2] - 500 - 2 * 1000, center[2] + 500 +
    2 * 1000))
```

Next we obtain coordinates of cell centers and x and y coordinates for the grid lines within small extent:

```
cells_centers <- coordinates(crop(dem, extSmall))
lines_x <- unique(c(cells_centers[, 1] - 500, cells_centers[,
```

```
    1] + 500))
lines_y <- unique(c(cells_centers[, 2] - 500, cells_centers[,
    2] + 500))
```

Now we obtain a subset of buildings shapefile and raster limited to the small extent:

```
fewBuildings <- crop(Buildings, extSmall)
fewBuildings.r <- crop(sum.buildings, extSmall)
```

Finally we plot elevation map of Switzerland with cantonal borders and arrow pointing to small extent shown as a blue square. We also plot buildings shapes of small extent with raster grid and values overlayed:

```
aux.arrow <- 1e+05  #to help with plotting arrow showing the small extent in full map
par(mar = c(0.2, 0.2, 0.2, 0.2), mfrow = c(1, 2))
plot(dem, legend = F, axes = F)
plot(ch.sp, add = T)
plot(extSmall, col = "blue", add = T)
arrows(x0 = center[1] + aux.arrow, y0 = center[2] + aux.arrow,
    x1 = center[1], y1 = center[2], col = "blue")
plot(crop(dem, extSmall), col = "white", legend = F, axes = F)
plot(fewBuildings, add = T, col = "grey")
abline(v = lines_x, h = lines_y, col = "red")
text(cells_centers, labels = values(fewBuildings.r), cex = 1,
    col = "blue")
```

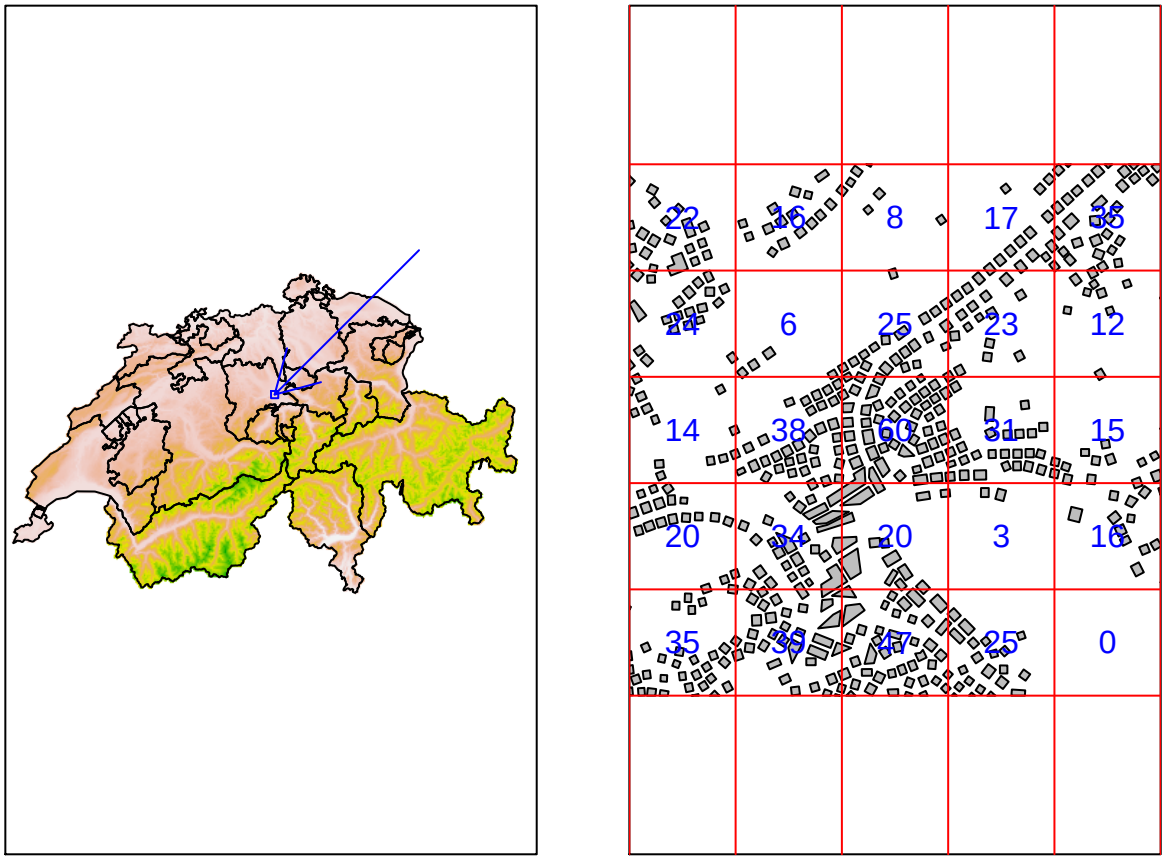

Figure 35: On the left a map of elevationwith cantonal borders and arrow pointing to small extent shown as a blue square. On the right a map of buildings (shapefile) with sum of buildings raster values and grid displayed: zoom in for 5000m x 5000m extent centered around cell with most buildings in Switzerland raster grid.

Again we can see that the *rasterization* does what we hoped it would do.

3.2.2 Buildings buffer

For the buildings buffer we work with `buffer` and rasters instead of`gBuffer` and SpatialPolygons because there are 313,670 building polygons so processing time is too long. The `buffer` function calculates a buffer of given *width* around all cells from raster given as first argument that are not NA.

The sum of buildings raster currently has zeros where there are no buildings. We need to replace those zeros with NA since `buffer` calculates buffer around cells that are not NA.

First we replace all NAs with zeros::

```
sum.buildings2 <- sum.buildings
sum.buildings2[which(values(sum.buildings2) == 0)] <- NA
```

Now we create a 1km buffer for buildings:

```
buildings.buffer <- buffer(sum.buildings2, width = 1000)
```

We then need to re-assign zero values to all cells that have NA values for plotting purposes. However cells that are outside of Switzerland should still have NA values.

Next we assign zero values to all NA valued cells and then re-assign NA values to those cells that fall outside of Switzerland using mask function and the elevation raster as a template:

```
buildings.buffer[which(is.na(values(buildings.buffer)))] <- 0
buildings.buffer <- mask(buildings.buffer, dem)
# summary(values(buildings.buffer))
```

Finally we plot buildings buffer and buildings rasters:

```
par(mar = c(0, 0, 0, 0))
plot(buildings.buffer, legend = F, axes = F, box = F)
plot(sum.buildings2, add = T, col = "yellow", legend = F)
legend(7e+05, 85000, legend = c("buildings", "buildings buffer"),
    fill = c("yellow", "green"))
```

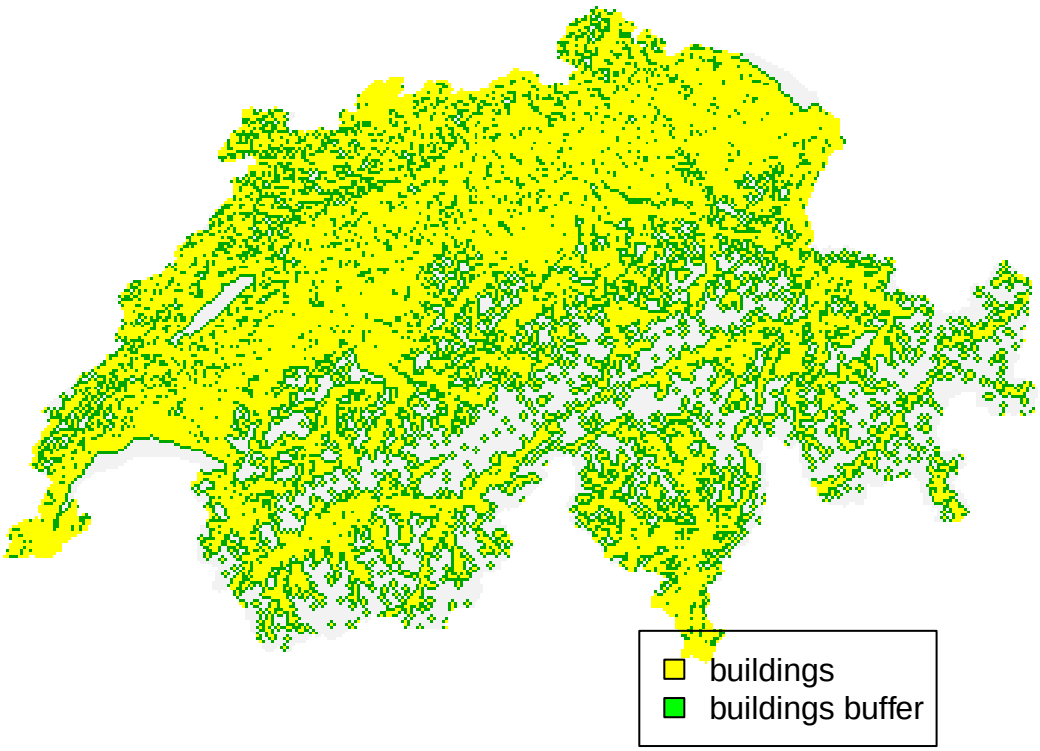

Figure 36: Map of Switzerland with locations where buildings are in yellow and locations where there are no buildings but that are within 1000m of buildings in green.

3.2.3 Parks

We want to create a new reserve at a location where there is currently none so we need to load a map indicating current park locations.

We will load shape files of national (nat_park) and regional (Parke) parks in Switzerland, convert them into rasters and combine them into a general parks raster indicating the location of parks in Switzerland. The national and regional park shapefiles were obtained from the Swiss Federal Office of Topography (swisstopo).

First we oad both park shape files and convert to raster using rasterize function:

```
# read regional parks shape file
Parke <- readOGR(dsn = "../data/Parke", layer = "Parke", p4s = prj)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/Parke", layer: "Parke"
## with 19 features
## It has 9 fields
```

```
Parke.r <- rasterize(x = Parke, y = dem, field = rep(1, length(Parke)))
# read national parks shape file
nat_park <- readOGR(dsn = "../data/nat_park", layer = "nat_park",
    p4s = prj)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "../data/nat_park", layer: "nat_park"
## with 2 features
## It has 9 fields
```

```
nat_park.r <- rasterize(x = nat_park, y = dem, field = rep(1,
    length(nat_park)))
```

Now we merge both national and regional park rasters into a parks raster:

```
parks.r <- Parke.r | nat_park.r
```

Finally we plot elevation rasters with parks overlayed:

```
par(mar = c(0, 0, 0, 0))
plot(dem, axes = F, box = F)
plot(parks.r, col = "green", add = T, legend = F)
```

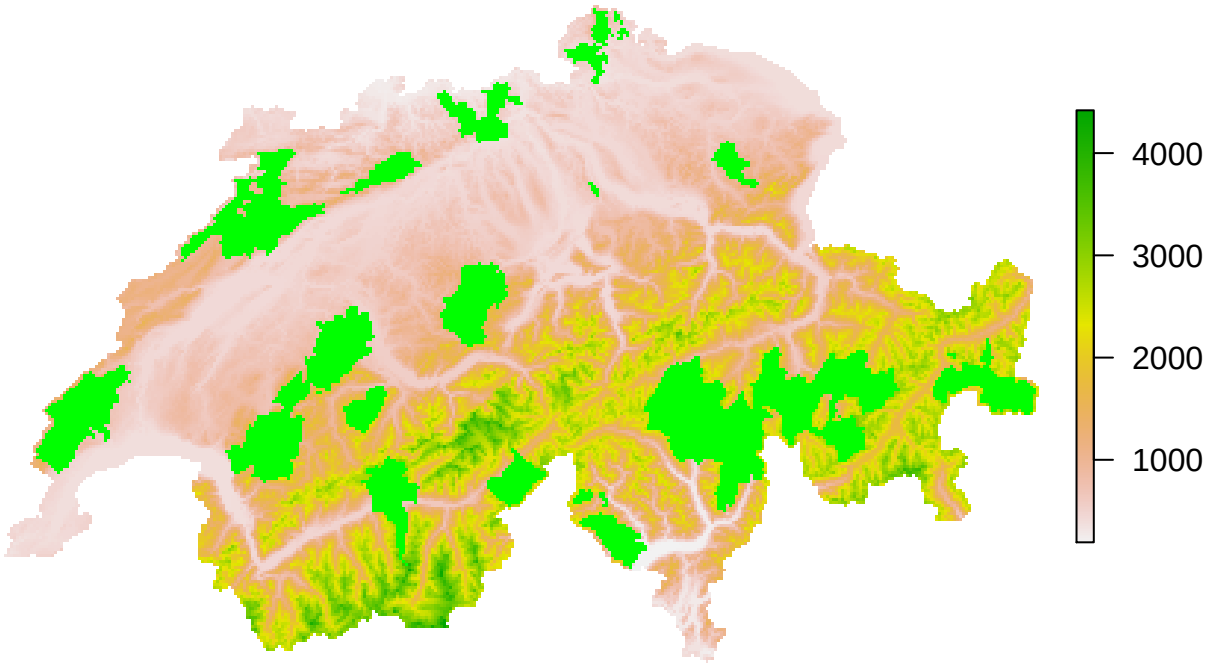

Figure 37: Map of elevation (raster) in Switzerland with national parks (raster) overlayed in green.

3.2.4 Potential reserve sites

We will now identify potential reserve locations.

First we create a raster identifying locations with a *high* bird species richness: 15 or more bird species:

```
# summary(values(BirdRich)) hist(values(BirdRich))
bird_high <- BirdRich > 15
```

All but one of the reserve criteria can be evaluated on a cell by cell basis: mininum distance to building, no lakes or parks and high bird diversity. We will identify cells that meet these criteria by using logical operators on the parks, lakes, buildings buffer and bird richness projection rasters. For criterion based on size of the park we will have to check the number of cells to which a given cell, meeting previous criteria, is connected to.

Next we prepare lakes and parks raster making sure locations with lakes (parks) have a value of 1/TRUE, locations without lakes (parks) have a value of 0/FALSE and locations outside of Switzerland have a value of NA:

```
Lakes.r[which(is.na(values(Lakes.r)))] <- 0
Lakes.r <- mask(Lakes.r, dem)
parks.r[which(is.na(values(parks.r)))] <- FALSE
parks.r <- mask(parks.r, dem)
# summary(values(buildings.buffer)) summary(values(Lakes.r))
# summary(values(parks.r)) summary(values(bird_high))
```

Now we create a raster indicating potential reserve cells: those with high richness not falling in lakes, existing parks and or near buildings (within buildings buffer):

```
Potential <- !buildings.buffer & !Lakes.r & !parks.r & bird_high
# summary(values(Potential))
```

Finally we plot *potential* raster showing cells which meet criteria 1-4:

```
par(mar = c(0, 0, 0, 0))
plot(Potential, axes = F, box = F, col = c("grey", "green"))
```

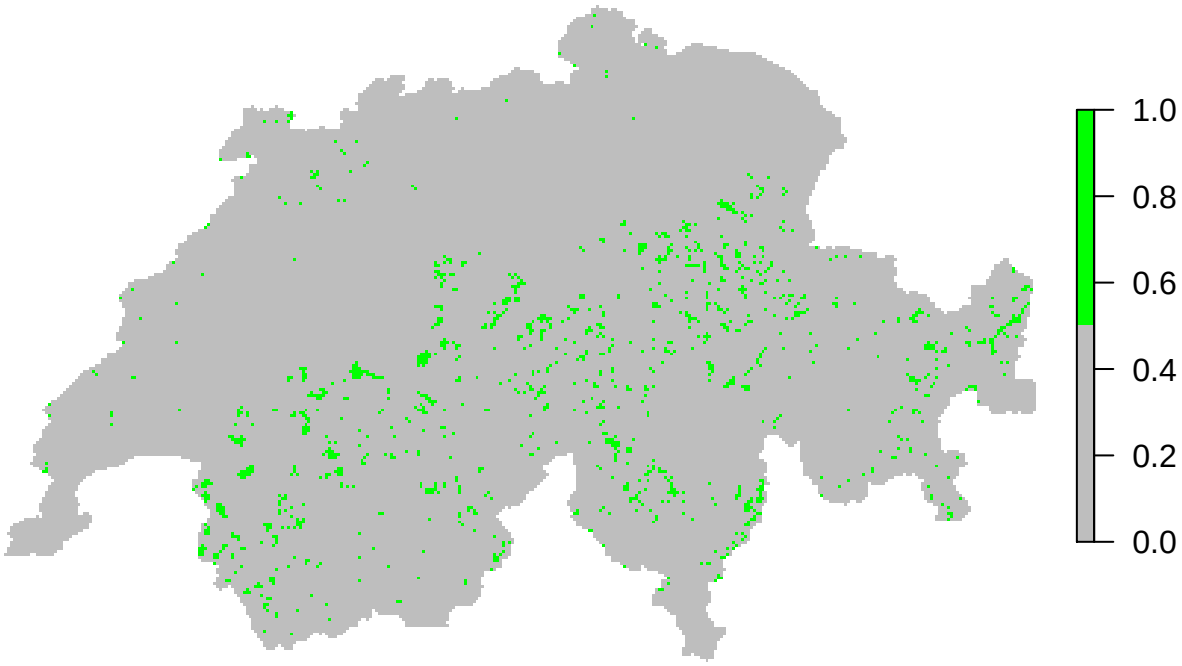

Figure 38: Map indicating potential reserve sites (raster). Cells in green satisfy all requirements for new reserve location (except reserve area requirement for which number of contiguous cells meeting requirements has to be evaluated).

Are any of these spots at least 40km^2? The function `clump` from the R package `raster` detects patches of connected cells in the raster given as argument, giving each cell in the patch the same id, which is now the value of the resulting raster. Two cells are connected if you can get from one to the other with a series of one-cell movements to neighboring cells (8 surrounding cells) without ever arriving at an NA valued cell. We can then make a table of id frequencies and replace the id value of the raster with its frequency. We do this with the `freq` function from the R package `raster` which constructs a frequency table of the values of the raster passed as argument.

We will use the function `gCentroid` from the R package `rgeos` to obtain the centroid of each canton. This function takes as argument a shape object and calculates its centroid. We can then place the label of the canton name at the centroid of each canton in order to help us see where our potential reserve may be located.

First we identify patches of connected cells using clump function:

```
Potential.patches <- clump(Potential)
```

```
## Loading required namespace: igraph
```

We will now modify raster, using `freq` function, so that the value of each cell is the size of the patch it belongs to.

Now we obtain a table of the frequency with which each patch-ID, identifying a different patch of connected cells, appears. Exclude NA frequency from table:

```
tab <- freq(Potential.patches)
tab <- tab[1:(nrow(tab) - 1), ]
```

Next we assign raster cell value by matching the patch-ID value of the cell with the frequency of the ID :

```
Potential.patches[1:length(values(Potential.patches))] <- tab[match(values(Potential.patches),
    tab[, 1]), 2]
```

Now we prepare potential patches raster for plotting by first assigning NA valued-cells the value 0 and then cells outside of Switzerland the value NA:

```
Potential.patches[which(is.na(values(Potential.patches)))] <- 0
Potential.patches <- mask(Potential.patches, dem)
```

Finally we plot potential patches raster showing size of each patch and print a summary of the size of patches:

```
par(mar = c(0, 0, 0, 0))
plot(Potential.patches, axes = F, box = F)
plot(ch.sp, add = T)
text(coordinates(gCentroid(ch.sp, byid = T)), ch.sp$NAME_1, cex = 0.5)
```

Figure 39: Map indicating potential reserve sites (raster). Colored cells satisfy all requirements for new reserve location (except reserve area requirement). Color of cells indicates the number of cells that are connected to it. Includes cantonal borders.

```
pander(summary(values(Potential.patches)), caption = "Summary of areas of potential sites")
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
0	0	0	0.221	0	32	40886

Table 13: Summary of areas of potential sites

We see that of the sites that satisfy all the conditions the one with the largest area, located in the canton of Bern, has only 32 km^2. Our strategy will now be to remove the buildings buffer condition and only require that there not be any buildings on the potential reserve site. This will allow us to require a higher bird species richness value for the reserve site.

Now we redefine a potential reserve cell as a cell without buildings, lakes and parks, and with more than 35 bird species richness:

```
Potential2 <- !(sum.buildings > 0) & !Lakes.r & !parks.r & (BirdRich >
    35)
# summary(values(Potential2))
par(mar = c(0, 0, 0, 0))
plot(Potential2, axes = F, box = F, col = c("grey", "green"))
```

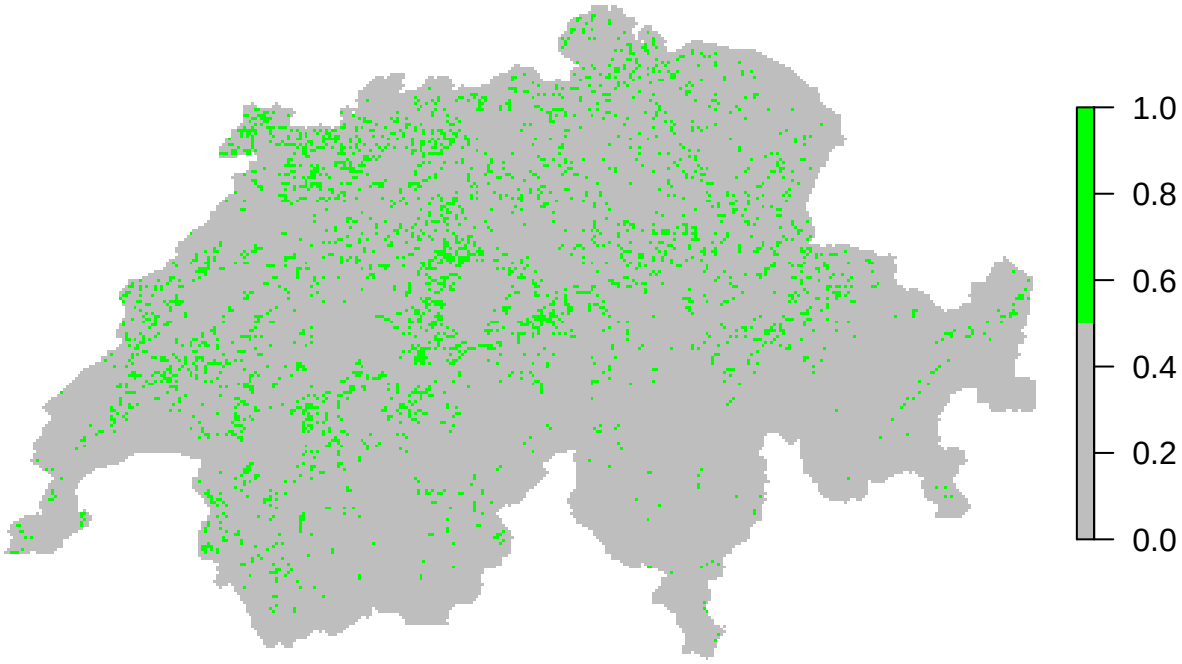

Figure 40: Map indicating potential reserve sites (raster). Cells in green satisfy all modified criteria for new reserve location (except reserve area requirement for which number of contiguous cells meeting requirements has to be evaluated).

Are any of these spots at least 40km^2?

We identify patches of connected cells using clump function and construct raster where the value of each cell is the size of the patch it belongs to:

```
Potential.patches2 <- clump(Potential2)
tab <- freq(Potential.patches2)
tab <- tab[1:(nrow(tab) - 1), ]
Potential.patches2[1:length(values(Potential.patches2))] <- tab[match(values(Potential.patches2),
    tab[, 1]), 2]
```

Now we prepare new potential reserve site raster for plotting as before and plot:

```
Potential.patches2[which(is.na(values(Potential.patches2)))] <- 0
Potential.patches2 <- mask(Potential.patches2, dem)

par(mar = c(0, 0, 0, 0))
plot(Potential.patches2, axes = F, box = F)
plot(ch.sp, add = T)
text(coordinates(gCentroid(ch.sp, byid = T)), ch.sp$NAME_1, cex = 0.5)
```

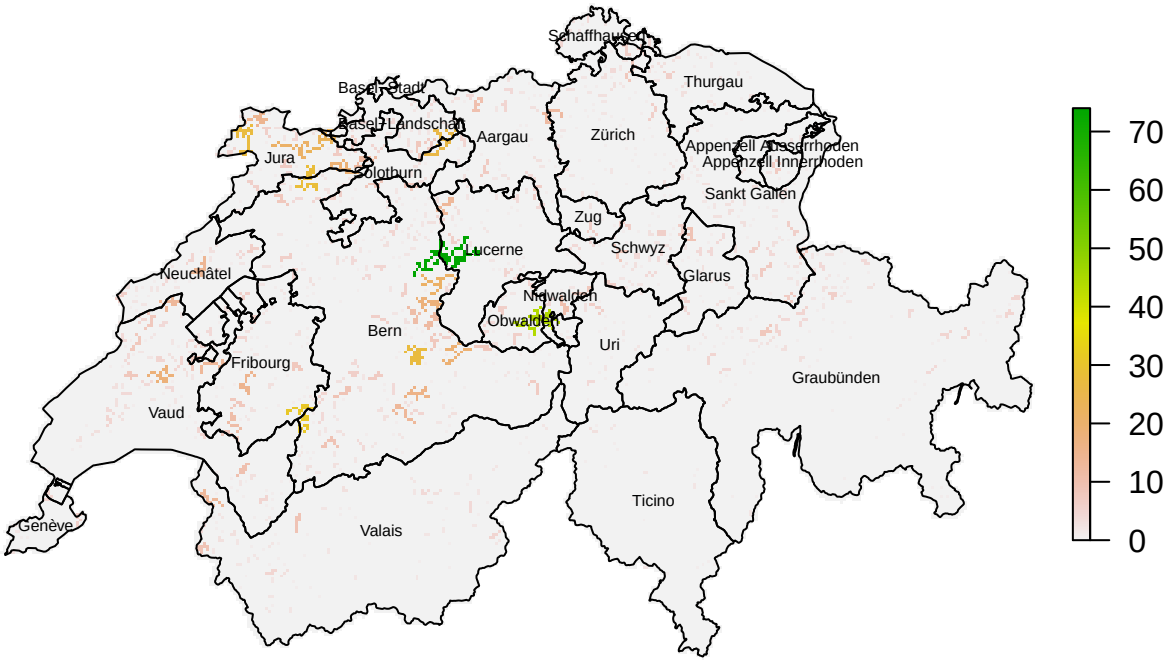

Figure 41: Map indicating potential reserve sites (raster). Colored cells satisfy all modified criteria for new reserve location (except reserve area requirement). Color of cells indicates the number of cells that are connected to it. Includes cantonal borders.

Finally we print a summary of the size of patches:

```
pander(summary(values(Potential.patches2)), caption = "Summary of areas of potential sites (relaxed cri
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
0	0	0	0.5667	0	74	40886

Table 14: Summary of areas of potential sites (relaxed criteria)

We see that, with these altered conditions, there is a potential site of 74km ^{2} on the border of the cantons of Bern and Lucerne.

3.2.5 Selection

We now obtain a shape object (polygon) representing the boundaries of our suggested new reserve park based on the patch of connected cells, identified above, which meet all the criteria. We will zoom in to the patch with most connected cells that meet all the criteria by obtaining the extent of these cells. Then we will plot the extent and use `drawPoly` function to draw a polygon that includes all these cells.

First we obtain coordinates of cells that belong to the potential site with maximum number of connected cells using xyFromCell function:

```
# plot only biggest patch, get coordinates of biggest patch
xyMaxPatch <- xyFromCell(Potential.patches2, which(values(Potential.patches2) ==
    max(values(Potential.patches2), na.rm = T)))
```

Next we create an extent from these coordinates using extent function:

```
extentMaxPatch <- extent(xyMaxPatch)
```

Now we plot a zoom-in of potential sites raster where limits of plot defined by the extent of the potential site with maximum number of connected cells:

```
plot(crop(Potential.patches2, extentMaxPatch))
```

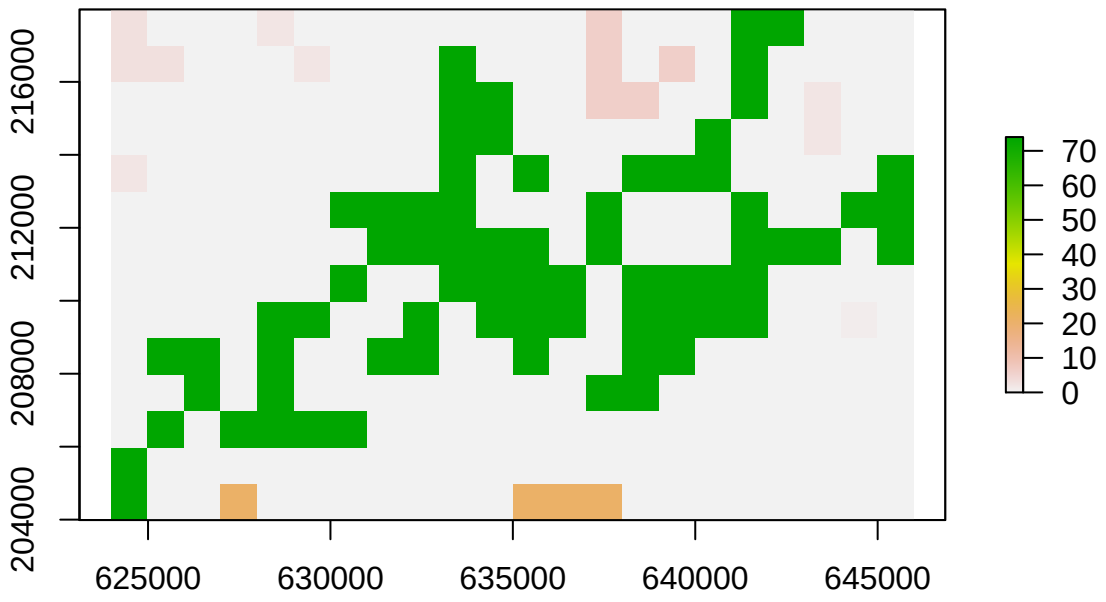

Figure 42: Largest potential site

Now we draw a polygon around the potential site with following commands:

```
myReserve <- drawPoly()
myReserve.r <- rasterize(myReserve, dem, field = 1)
writeRaster(myReserve.r, filename = "../data/myReserve/myReserve",
    format = "raster")
```

Finally we plot raster of selected reserve site together with existing parks raster:

```
par(mar = c(0, 0, 0, 0))
plot(myReserve, col = "green4", legend = F, box = F, axes = F)
plot(ch.sp, add = T)
plot(parks.r, add = T, col = c(NA, "green"), legend = F)
plot(water.r, add = T, col = "light blue", legend = F)
legend(730000, 90000, fill = c("green", "green4"), legend = c("existing park",
    "new reserve"))
```

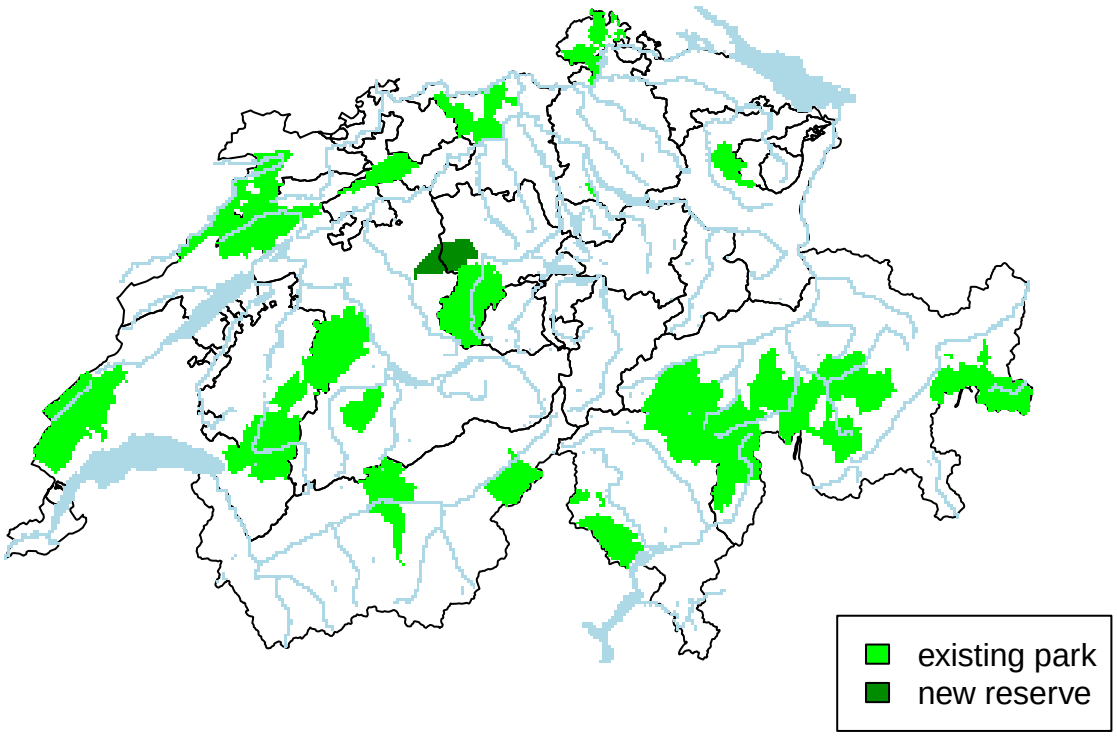

Figure 43: Map of water (raster), existing national parks (raster), cantonal borders (shapefile) and new reserve location (raster).

Finally:

- Compare your potential park to the existing parks in Switzerland (use topographic map and parks rasters). Suggest minimum two places where it would be interesting to build a reserve for birds. Take into account that this new park should cover a region of Swtzerland where no such park exists. Identify the region where the park would be situated and some features of the regions. Would this region be suitable for a nature regional park?
- Think of other organisms (plants, mammals, reptiles, etc.) and try to find sites that could also be interesting for them (go to https://map.geo.admin.ch/ for some help).

- Draw the final shapefile of the park based on the features from the topographic map, for instance accounting for position of large roads or other boundaries, or natural barriers such as mountain ranges.
- Create a map of Switzerland with the location of existing parks in one color and the new potential park in another.
- Think about the limitation of using species richness to design such a park and propose another approach to improve the procedure.

4 Installing rgeos and rgdal packages on MacOSX

Instructions for installing rgeos and rgdal packages on MacOSX:

- install xquartz, see http://www.xquartz.org/
- install GEOS and GDAL frameworks, see http://www.kyngchaos.com/files/software/frameworks/GDAL_Complete-1.11.dmg
- optionally XCode and Apple Command Line Developer Tools, see http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/
- optionally Fortran compiler, see http://stat.ethz.ch/CRAN/doc/manuals/r-release/R-admin.html#OS-X