Advanced spatial R workshop

Tobias Andermann

9/27/2019

Accessing biodiveristy data through web services (IUCN)

Dependencies:

```
library(raster)
library(sf)
library(ncf)
library(spdep)
```

In this exercise we are going to test which predictors are spatially significantly correlated with a biological variable. For example you would apply the tools we are covering in this workshop if you want to test if species diversity in your studied organism group is significantly correlated with temperature, temperature, predator diversity, etc. While this is a common operation in particular in marco-ecological/-evolutionary studies, there are several caveats that one needs to account for. Here we will deal with the main ones, which are namely raster projection, testing for and accounting for spatial autocorrelation and applying different models (namely GLM and SAR) for statistical testing.

In this scenario we will check which environmental predictors have a significant influence on mammal species diversity, i.e. what is driving hgiher or lower mammal diversity (e.g. do warmer areas have higher diversity than colder ones, wetter than drier, …)

The response variable

In our example mammal species diversity is the response variable, i.e. we want to figure out how species diversity responds to different predictors. You will need to produce the species diversity raster by adding up the presence/absence rasters belonging to each of the species of your mammal group of choice.

For selecting the species names belonging to your chosen group (order, family or genus), you can use the following taxonomy information available from the Phylacine database (https://megapast2future.github.io/PHYLACINE_1.2/):

```
mammal_taxonomic_data = read.csv('../data/Data/Traits/Trait_data.csv')
```

For example you can extract all species from the order `Carnivora` like this:

```
species_list = mammal_taxonomic_data[mammal_taxonomic_data$Order.1.2 == 'Carni
vora',]$Binomial.1.2
```

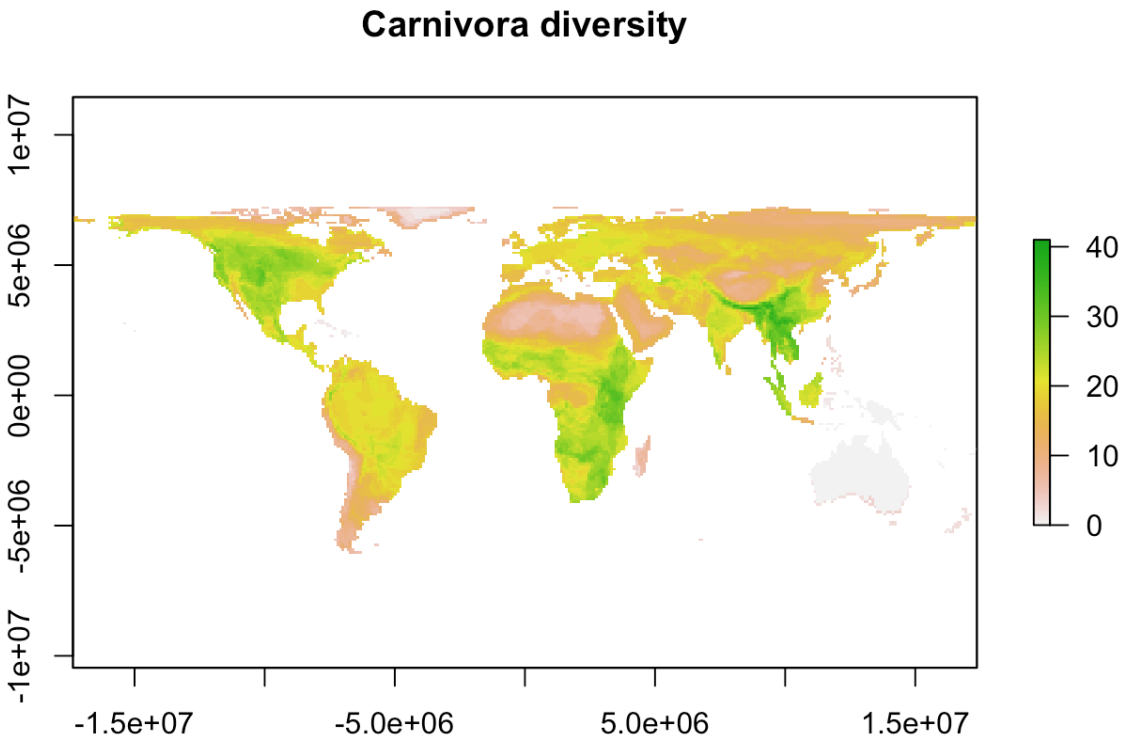
Now load the range files for these species and add them up for producing an overall global diversity

map of the selected group. Also load a shape of a world map to crop out only land cells (using the `mask()` function). This is exactly what we were doing in tutorial 2 of the first day (http://htmlpreview.github.com/?https://github.com/tobiashofmann88/spatial_R_course/blob/master/tutorials/tutorial_2.html). Try to solve this yourself, but you can look at the code below for help:

```
library(raster)
library(sf)

range_folder = '../data/Data/Ranges/Present_natural/'
species_ranges=list.files(range_folder,pattern = '*.tif')

# initiate the raster
raster = raster(paste0(range_folder,species_ranges[1]))
raster[raster>0] = 0

for (i in species_list){
  index = which(grepl(i, species_ranges))
  if (length(index)==0){
    no_match=TRUE
  }else{
  species_raster=raster(paste0(range_folder,'/',i,'.tif'))
  raster = raster+species_raster
  }
}

world_map = st_read('../data/global/ne_110m_land/ne_110m_land.shp')
```

```
## Reading layer `ne_110m_land' from data source `/Users/tobias/GitHub/spatial
_R_course/data/global/ne_110m_land/ne_110m_land.shp' using driver `ESRI Shapef
ile'
## Simple feature collection with 127 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: -180 ymin: -90 xmax: 180 ymax: 83.64513
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
transformed_world = st_transform(world_map,projection(raster))
world_spatial <- as(transformed_world, 'Spatial')
diversity_raster = mask(raster,world_spatial)
plot(diversity_raster,main='Carnivora diversity')
```



To get a better understanding of your species diversity raster, just print the raster object to screen :

```
## class       : RasterLayer
## dimensions  : 142, 360, 51120  (nrow, ncol, ncell)
## resolution  : 96486.27, 96514.96  (x, y)
## extent      : -17367529, 17367529, -6356742, 7348382  (xmin, xmax, ymin, ym
ax)
## coord. ref. : +proj=cea +lon_0=0 +lat_ts=30 +x_0=0 +y_0=0 +datum=WGS84 +uni
ts=m +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0, 41  (min, max)
```

Check the dimension of you raster using the `dim()` command, which tells you how many cells the raster contains along the x and y axis.

```
dim(diversity_raster)
```

```
## [1] 142 360   1
```

And the resolution of the raster:

```
res(diversity_raster)
```

```
## [1] 96486.27 96514.96
```

The resoluiton is scaled in meters, so this means that the average cell size is ~96x96km i this raster. However, this is just the average size, and the cell sizes between cells at different parts of the globe differ in their actual size. As mentione din the lecture, this average cell size in the Behrmann projection (which is the projection of the raster) is only exactly manifested at 30 degrees north and 30 degrees south.

You can get an idea of the coordinate system your raster data is stored in by checking the extent of the raster:

```
extent(diversity_raster)
```

```
## class       : Extent
## xmin        : -17367529
## xmax        : 17367529
## ymin        : -6356742
## ymax        : 7348382
```

This is how you check the projection of the cluster:

```
projection(diversity_raster)
```

```
## [1] "+proj=cea +lon_0=0 +lat_ts=30 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_
defs +ellps=WGS84 +towgs84=0,0,0"
```

This projection is referred to as Behrman format.

Predictor data

Now it's time to load the data for the predictor variables, which are the factors we want to test for correlation with mammalian species diversity. You can download whatever variables you want to check (for example get annual precitipitation data if you want to test if temperature is a predictor of species diversity). A lot of useful climatic data can be found at http://chelsa-climate.org/bioclim/ (http://chelsa-climate.org/bioclim/), but feel free to use any data source you want to. In this example I will work with annual average temperature grid data, but you should preferably chose a different variable to test:

```
temperature_file = '../data/CHELSA_bio10_01.tif'
temperature_raster = raster(temperature_file)
#plot(temperature_raster,main='temperature')
temperature_raster
```

```
## class       : RasterLayer
## dimensions  : 20880, 43200, 902016000  (nrow, ncol, ncell)
## resolution  : 0.008333333, 0.008333333  (x, y)
## extent      : -180.0001, 179.9999, -90.00014, 83.99986  (xmin, xmax, ymin,
ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,
0,0
## data source : /Users/tobias/GitHub/spatial_R_course/data/CHELSA_bio10_01.ti
f
## names       : CHELSA_bio10_01
## values      : -32768, 32767  (min, max)
```

If you are using bioclim data you'll see that the resolution of the raster is incredibly high, which makes for very slow processing times for all operations on the raster. Let's therefore first reduce the data to a lower resolution.

If you want to rescale your raster to a coarser resolution you can use the `aggregate()` function. Say you want to change the resolution of your raster to roughly 0.1 grid size (approx. 10x10km cell-size at the equator), you can rescale the raster by factor `0.1/res(temperature_raster)`. Note that the aggregate function only accepts integer values. If a factor is provided that is not an integer but a float, the function will round it to the closest integer value. This command will take around 10 minutes to finish.

```
# round the values of the extent of this raster to integers
extent(temperature_raster) = round(extent(temperature_raster))
temperature_raster_reduced = aggregate(temperature_raster, fact = 0.1/res(temp
erature_raster))
temperature_raster_reduced
```

```
## class       : RasterLayer
## dimensions  : 1740, 3600, 6264000  (nrow, ncol, ncell)
## resolution  : 0.1, 0.1  (x, y)
## extent      : -180, 180, -90, 84  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,
0,0
## data source : /private/var/folders/13/rrlwnj8d3fn76tbz9yfcv0kr0000gn/T/Rtmp
oIbqgU/raster/r_tmp_2020-05-27_010715_94554_02501.grd
## names       : CHELSA_bio10_01
## values      : -525, 335.6042  (min, max)
```

Compare the difference in dimensions between the original raster and the reduced one.

```
dim(temperature_raster)
```

```
## [1] 20880 43200     1
```

```
dim(temperature_raster_reduced)
```

```
## [1] 1740 3600    1
```

We need to make sure that our predictor data (temperature) is in the same projection as the response variable data (species diversity).

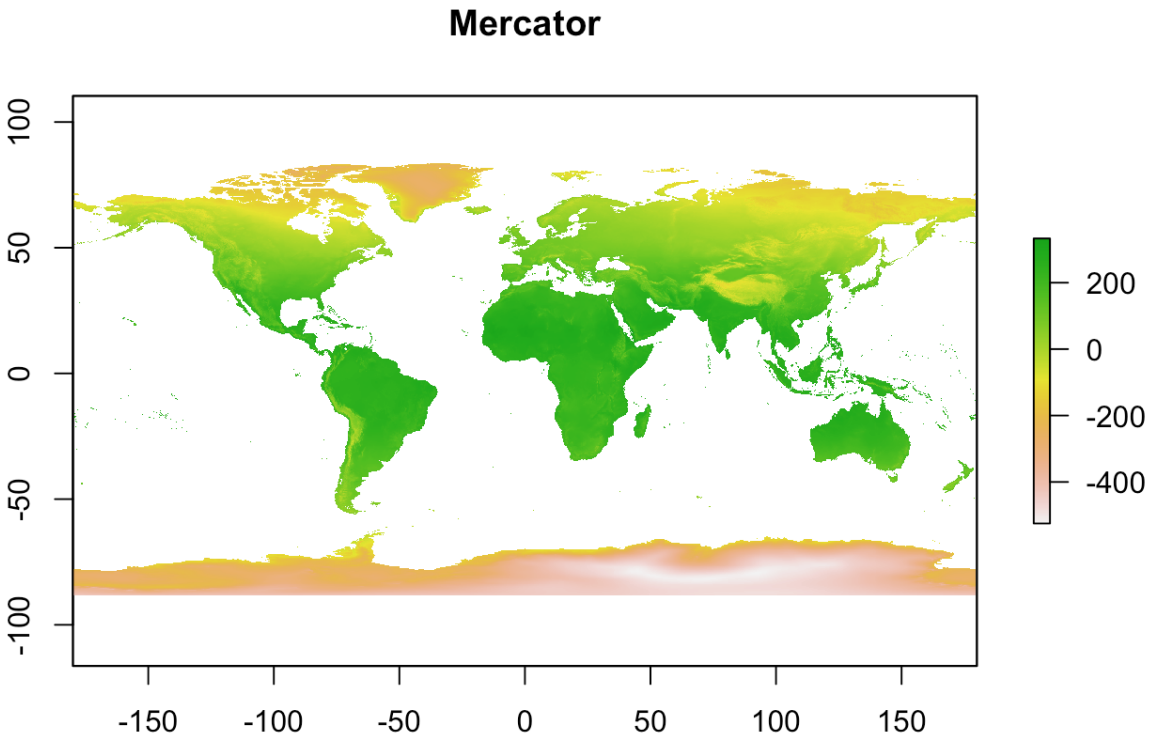
It makes most sense to project all data into a CEA projection, since this will optimize the cells towards having a close to equal area. If we wouldn't optimize for equal area but instead use the Mercator projection (lon-lat), our species diversity values would be biased toward the equator where cells would be bigger than toward the poles and would therefore inflate the values of species diversity the bigger the cell is. Using a CEA projection instead minimizes this bias. However some bias remains as it is impossible to project a globe into perfectly equal sized grid cells. This will take a few minutes to run.

```
temperature_data_cea = projectRaster(temperature_raster_reduced,crs="+proj=cea
+datum=WGS84 +lat_ts=30")
```

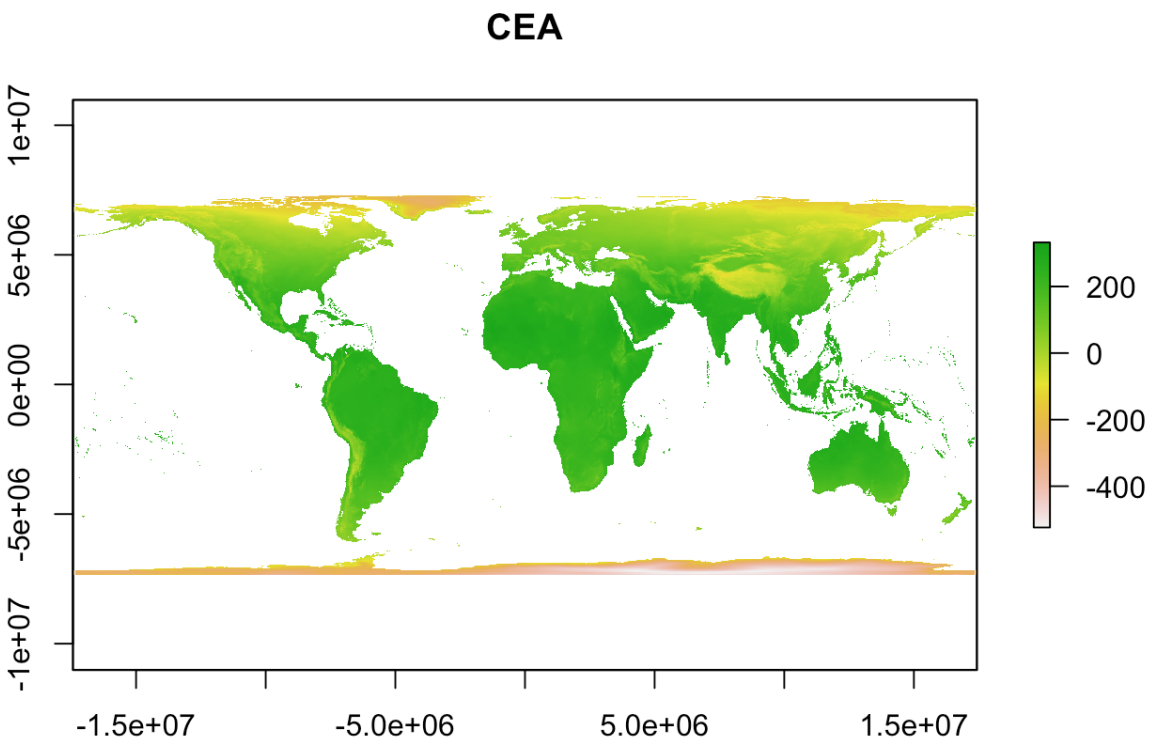
To remind you what we just did: - we loaded the raw climate data raster, which had a very high resolution - we reduced the resolution of the raster with `aggregate()` to roughly have 10x10km grid size - we projected the raster into the CEA projection, using `projectRaster()`

To demonstrate the difference between the two projections, plot the raster in both projections and compare e.g. the size of Greenland in relation to areas around the equator:

```
plot(temperature_raster_reduced,main='Mercator')
```



```
plot(temperature_data_cea,main='CEA')
```



Also, notice how the resolution of your raster has changed too by transforming it into another projection. This is expected because the cells of the raster are being restructured and altered in size when changing from one projection into another.

```
print(dim(temperature_raster_reduced))
```

```
## [1] 1740 3600    1
```

```
print(dim(temperature_data_cea))
```

```
## [1] 1163 3613    1
```

Matching the spatial data

Now we need to make sure that we have the exact same number of cells (rows and columns) for our response variable and predictor variables, in order to be able to match each spatial biodiversity value with a predictor variable value.

```
print(dim(diversity_raster))
```

```
## [1] 142 360   1
```

```
print(dim(temperature_data_cea))
```

```
## [1] 1163 3613    1
```

We can use the `resample()` function to match the dimensions of one raster to those of another. However, since our `diversity_raster` has a much lower resolution, and we would be producing spatial biases when just resampling values from our higher resolution raster based on the dimensions of this lower resolution raster, it is good practice to simply increase the dimensions of the reference raster (`diversity_raster` in our case) by dividing existing cells into multiple cells with the same exact value using `disaggregate()` to roughly match the dimensions of the target raster (`temperature_data_cea`). Therefore we will first inflate the number of cells in the `diversity_raster` by factor 10, then match the dimensions of the temperature raster to that of the inflated diversity raster with `resample()` and then reduce the resolution of the resampled temperature raster by factor 10 (with `aggregate()`) to get it to exactly match the original diversity raster. If this doens't make sense to you have a look at the slides about the resampling process and feel free to ask us if this is still unclear.

```
diversity_raster_dis = disaggregate(diversity_raster,10)
temperature_data_matched_10 = resample(temperature_data_cea, diversity_raster_
dis, method='bilinear')
temperature_data_matched = aggregate(temperature_data_matched_10, 10)
```

Let's see if it worked:

```
print(dim(diversity_raster))
```

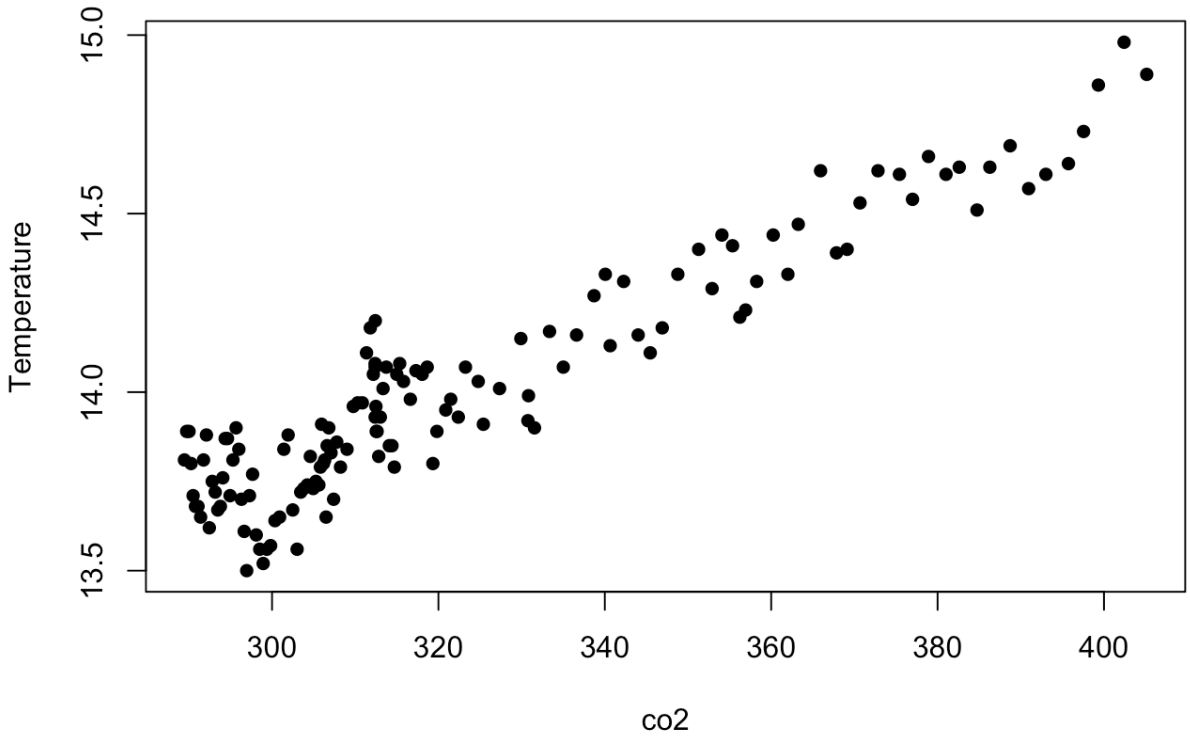
```
## [1] 142 360   1
```

```
print(dim(temperature_data_matched))
```

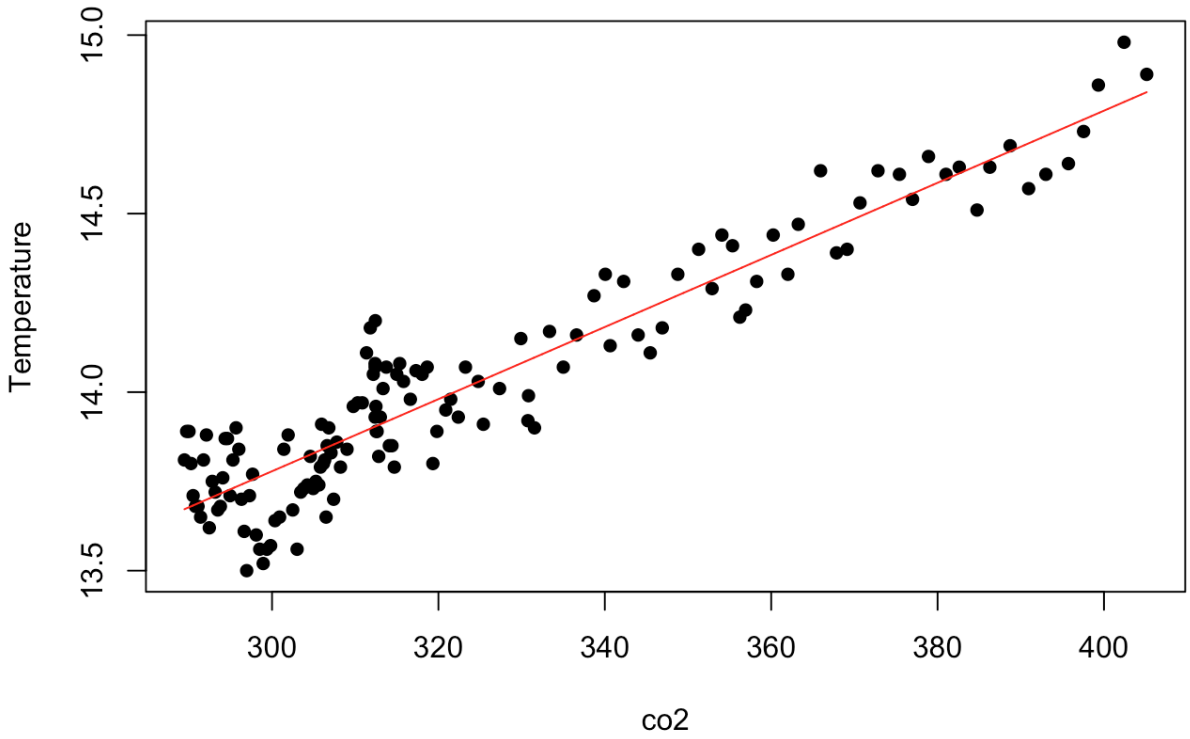
```
## [1] 142 360   1
```

Fitting linear models

```
global_temp = read.csv('../data/global_temp.txt',sep='\t')
co2 = read.csv('../data/co2.txt',sep='\t')
plot(co2$co2,global_temp$Temperature, pch = 16, xlab = "co2", ylab = "Temperat
ure")
```




```
combined = cbind(global_temp,co2)
model <- glm(Temperature ~ co2, family = gaussian ,data = combined)
xweight <- seq(range(co2$co2)[1], range(co2$co2)[2], 0.01)
yweight <- predict(model, list(co2 = xweight),type="response")
plot(co2$co2,global_temp$Temperature, pch = 16, xlab = "co2", ylab = "Temperat
ure")
lines(xweight, yweight,col='red')
```



Temporal auto-correlation

Disclaimer: Some of the following examples are borrowed from a great tutorial on spatial auto-correlation at https://rspatial.org/raster/analysis/3-spauto.html (https://rspatial.org/raster/analysis/3-spauto.html).

Before we get into spatial auto-correlation in particular, let's first talk about auto-correlation in general. If your data are auto-correlated, it means that they are not independent data points but that they are correlated to some extent. When using auto-correlated data without correcting for the correlation, one usually greatly overestimates the effective sample size.

For example if you want to measure a person's weight through time, you would expect two measurements which are close to each other in time to also be similar in the measured variable. To measure the degree of association over time, we can compute the correlation of each observation with the next observation.

Check for temporal auto-correlation. We are going to compute the "one-lag" auto-correlation, which means that we compare each value to its immediate neighbour, and not to other nearby values.

For CO2:

```
values <- co2$co2
a <- values[-length(values)]
b <- values[-1]
print(cor(a,b))
```

```
## [1] 0.99992
```

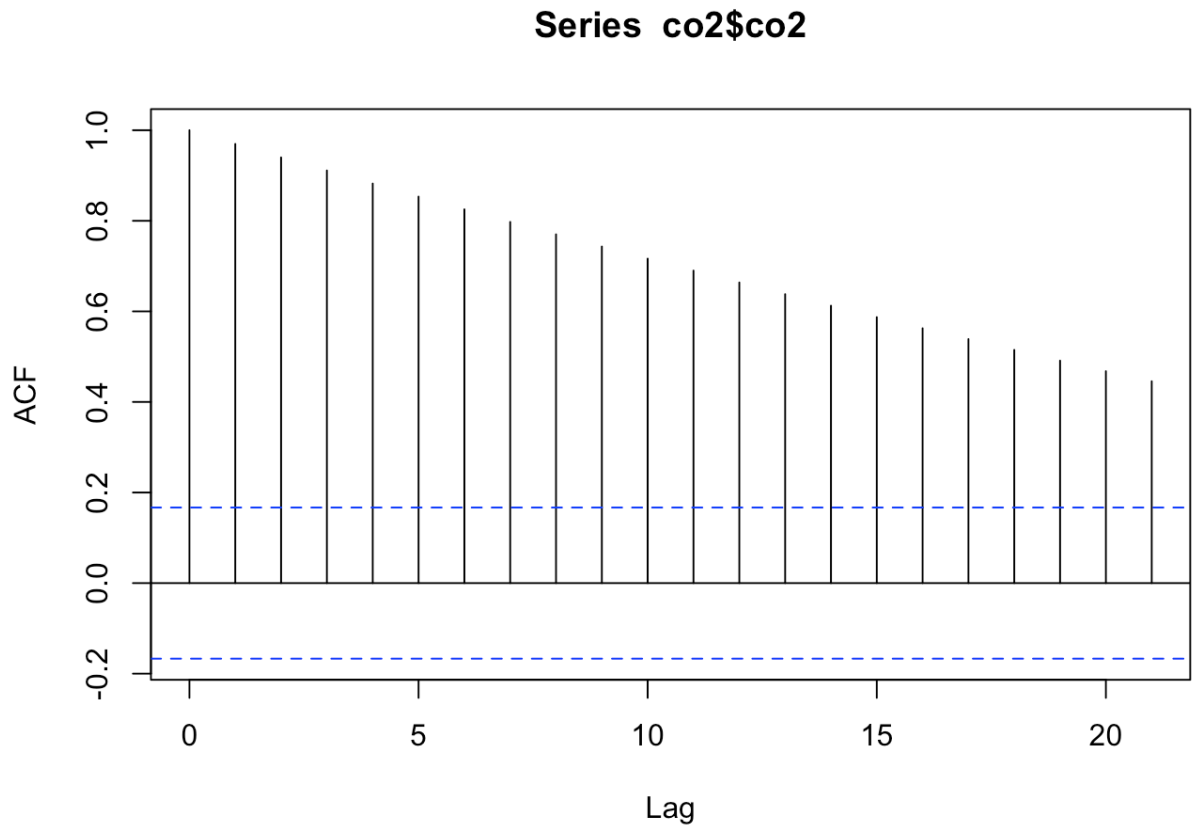
And for temperature:

```
values <- global_temp$Temperature
a <- values[-length(values)]
b <- values[-1]
cor(a,b)
```

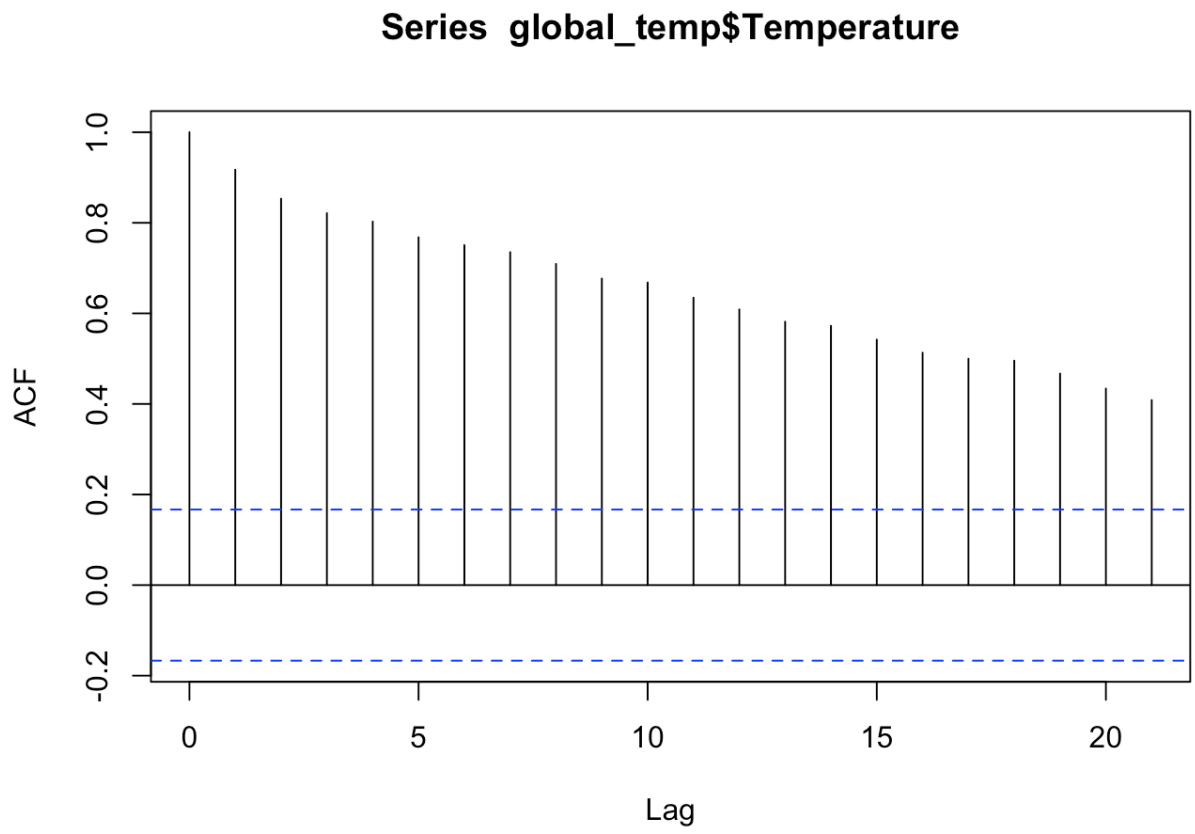
```
## [1] 0.9426816
```

These values indicate a very strong positive temporal auto-correlation for the “one-lag” method. There are also integrated and more elegant ways in R for determining auto-correlation than our manual implementation of the “one-lag” auto-correlation. For example the `acf()` function computes the autocorrelation for several lags:

```
acf(co2$co2)
```




```
acf(global_temp$Temperature)
```



We see that both global temperature and CO2 content of the atmosphere each show a strongly positive temporal auto-correlation. In this case it is caused by a clear temporal trend in the data, since CO2 levels are increasing with time and the temparature in reponse increases as well. However, temporal auto-correlation could also occur if data points of “neighbouring” years influence each other, without there being an overall trend.

Spatial auto-correlation

Similar to temporal auto-correlation, spatial auto-correlation means that two points or raster cells that are close to each other in space have similar values. In our example, in case our raster cells are spatially auto-correlated, we expect the species diversity values of two neighbouring cells to be more similar to each other than to further away cells (on average).

This auto-correlation can be exogenous (caused by some unknown/untested effects that effect neighbouring cells in a similar manner) or endogenous (caused by the variable we’re testing, e.g. temperature).

In the following we’re trying to quantify the degree to which neighbouring raster cells are similar to each other (more specifically we’re determining how similar their residuals (https://www.statisticshowto.datasciencecentral.com/residual/) are), using different definitions/thresholds of “neighbourhood”. We then include the determined autocorrelation into our model in order to account for the sum of the exogenous auto-correlation casued by unknown factors. Only by accounting for this can we measure the true effect of our tested variables on species diversity.

But before getting into neighbourhoods etc. let us first fit a general linear model to our species diversity data, to see if they correlate with our predictor variable (temperature), without worrying about spatial auto-correlation and neighbourhoods.

First we need to bring our raster data into a dataframe format. For this we extract the coordinates (raster cell centroids) using the `coordinates()` function. The coordinates should be identical for both of our rasters, check if that is the case by repeating the command below for your raster of predictor values. You should get the same coordinates and number of points for both rasters.

```
coordinates = coordinates(diversity_raster)
```

Now extract the corresponding values from both rasters, using the `values()` function. Then we put the coordinates and the values from both rasters together into one dataframe.

```
species_div = values(diversity_raster)
temperature = values(temperature_data_matched)
all_data_merged = as.data.frame(cbind(coordinates,species_div,temperature))
```

There are a lot of NA values resulting from the water cells in the two rasters, which didn't have any values assigned to them. We need to remove those before continuing to work with the data. Therefore let's just remove all rows from the dataframe that contain NA's using the `complete.cases()` function, which only extracts complete rows with data:

```
final_data = all_data_merged[complete.cases(all_data_merged),]
```

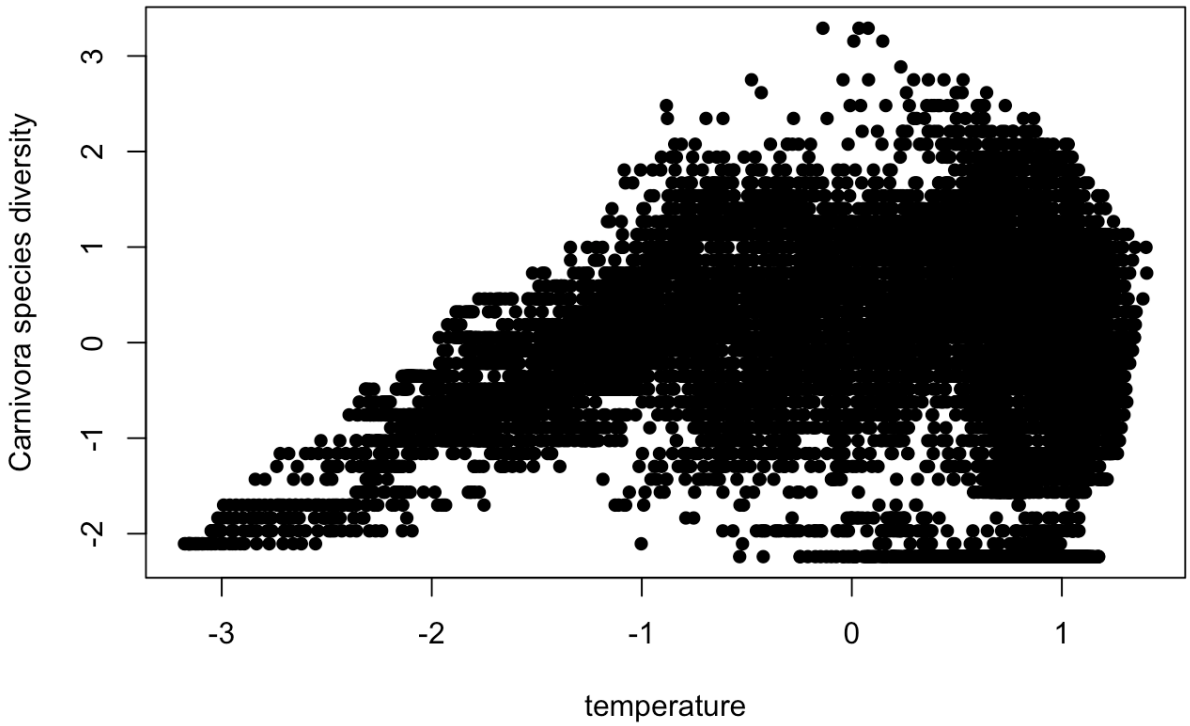
Further it is important to scale the values for both the response and the predictor variables to be centered in 0 using the `scale()` for our downstream statistical tests:

```
final_data$species_div=scale(final_data$species_div)[,1]
final_data$temperature=scale(final_data$temperature)[,1]
```

General linear model (GLM)

Now plot the final values of the predictor we want to test against the response variable to get a first impression on their relationship:

```
plot(final_data$temperature,final_data$species_div, pch = 16, xlab = "temperat
ure", ylab = "Carnivora species diversity")
```

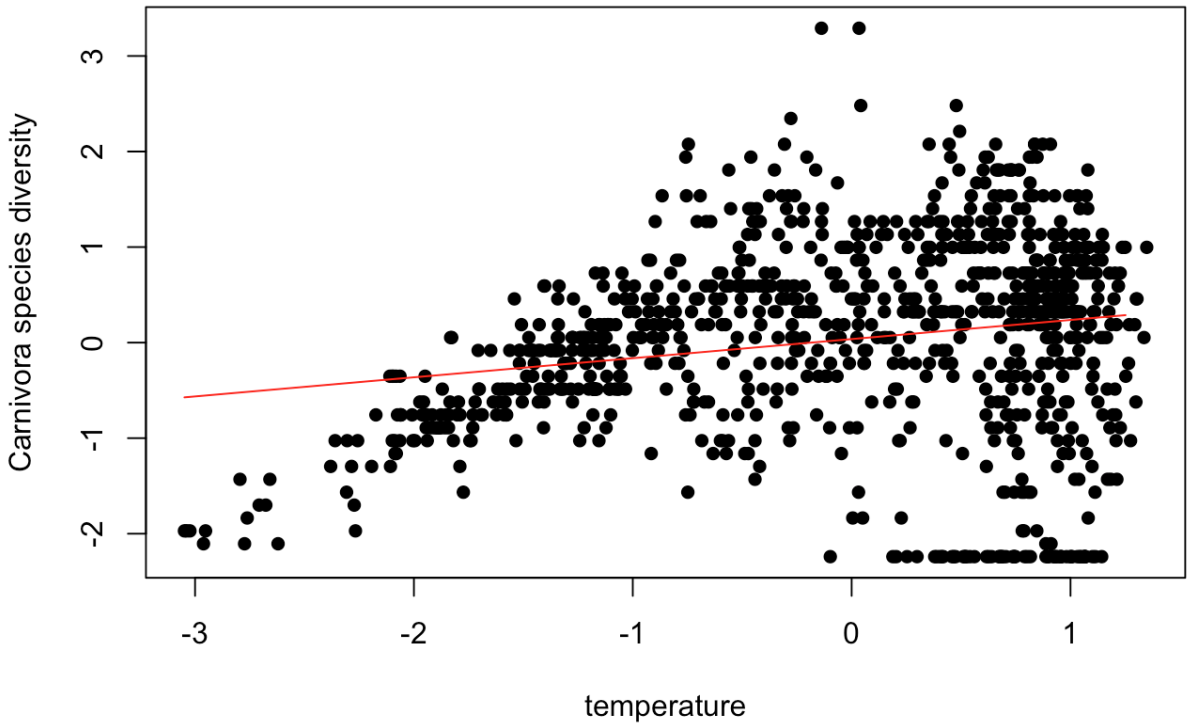


For faster computation, we'll go through the following steps just using a subsample of the data. Note that eventually you need to run the final model using all of the data, but for now let's stick to the subsample (we are setting a seed with the `set.seed()` command to make sure the same samples will be selected for the tutorial purpose. You won't need to set a seed).

```
set.seed(42)
# take a random sample of n points from the dataframe
subsample = final_data[sample(nrow(final_data), 1000), ]
```

Now fit the linear model to the data and plot the results:

```
glm_model = glm(species_div~temperature,data=subsample)
xweight = seq(range(subsample$temperature)[1], range(subsample$temperatur
e)[2], 0.1)
yweight = predict(glm_model, list(temperature = xweight),type="response")
plot(subsample$temperature,subsample$species_div, pch = 16, xlab = "temperatur
e", ylab = "Carnivora species diversity")
lines(xweight, yweight,col='red')
```



You can check how strong your tested predictor affects the response variable by using the `summary()` command on the model:

```
summary(glm_model)
```

```
##
## Call:
## glm(formula = species_div ~ temperature, data = subsample)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5058  -0.4574   0.1375   0.6152   3.2810
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03699    0.03168   1.168    0.243
## temperature  0.20009    0.03135   6.383 2.65e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9995017)
##
##     Null deviance: 1038.2  on 999  degrees of freedom
## Residual deviance:  997.5  on 998  degrees of freedom
## AIC: 2841.4
##
## Number of Fisher Scoring iterations: 2
```

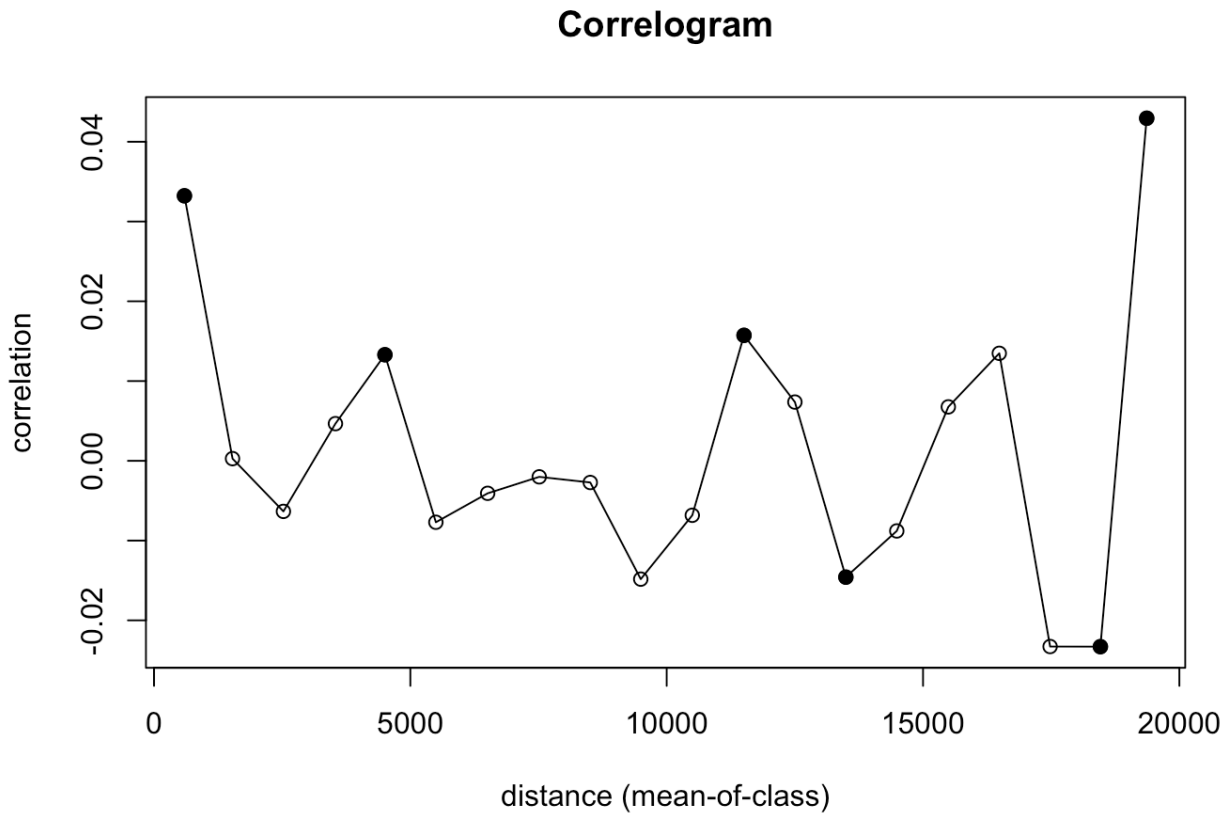
In the `Coefficients` section you can find the effect size of your predictor (value after the predictor variable name). Also you can see how significant the effect is by looking at the p-value (`Pr(>|t|)`). In our case it is highly significant.

Earlier in the temporal auto-correlation example we determined the auto-correlation at different lags using the `acf()` function. For spatial data we can use the `correlog()` function which calculates the Moran's I (as mentioned in the introduction slides) for different distances of points. This is a measure of the spatial auto-correlation of the residuals at different distances.

```
library(ncf)
autocorrelation_glm = correlog( subsample$x, subsample$y, glm_model$residuals,
increment=1000, latlon=T, resamp=100)
```

```
## 10  of  100
20  of  100
30  of  100
40  of  100
50  of  100
60  of  100
70  of  100
80  of  100
90  of  100
100  of  100
```

```
plot(autocorrelation_glm)
```



You can see that for close distances (< 1000 km) we find a stronger positive spatial auto-correlation. This is a common pattern for spatial data and is the reason why we need to bother with accounting for this auto-correlation in our model.

Linear models including neighbourhoods (SAR model)

In order to formalize this spatial auto-correlation and integrate it into our linear model, we need some measure of neighbourhood (i.e. a measure of what is considered a "nearby point"). First step is to transform our coordinates back from Behrman (CEA) into lat-lon format, since this is required to properly calculate the distance between cells. As was noted in the lecture CEA is area true but produces wrong distances between cells.

```
coordinates = cbind(subsample$x,subsample$y)
# define the current projection (Behrman)
coordinates_sp = SpatialPoints(coordinates,proj4string = CRS(projection(divers
ity_raster)))
# transform to lat-lon projection
coordinates_transformed = spTransform(coordinates_sp, CRS("+proj=longlat +datu
m=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"))
# turn back into data frame
coordinates_df = as.data.frame(coordinates_transformed)
subsample$x = coordinates_df$coords.x1
subsample$y = coordinates_df$coords.x2
```

Now we can use the `knearneigh()` function to identify the n closest neighbouring cells for each given cell. In the example below we extract the 2 closest neighbours for each cell (you can change the value to anything you like, e.g. you can extract the 10 closest neighbours instead). Note the argument `longlat = T` which tells the function that the input data is in lon-lat format:

```
library(spdep)
nearest_neighbours_2 = knearneigh(cbind(subsample$x, subsample$y),2,longlat =
T)
# This function sorts out the spatial elements
neighbourlist_2_closests = knn2nb(nearest_neighbours_2)
# This illustrates what we have done
plot(neighbourlist_2_closests, cbind(subsample$x, subsample$y))
```

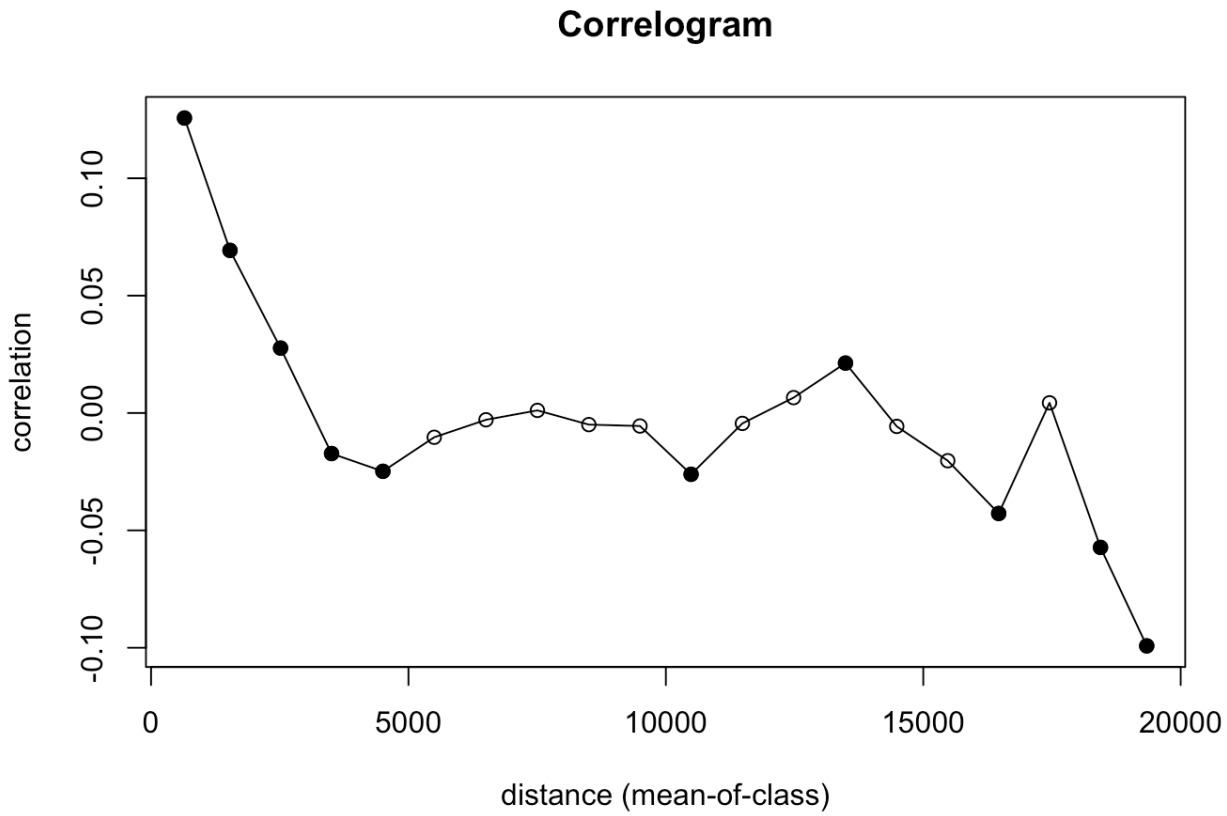
This looks very abstract, but you can roughly see the continents or islands the points in our random subsample were drawn from and the two closest neighbours of each point connected by lines.

Now we will apply the SAR model (`errorsarlm()`), which allows us to incorporate the neighbourhood of points. The formula of the SAR model is a modification of the general linear model (`y = X beta + u, u = lambda W u + e`), adding the error term `u` . The `W` in this error term are our modeled neighbourhood relationships, which are parsed to the function using the `listw()` argument.

```
sar_model = errorsarlm(species_div~temperature, data=subsample, listw=nb2listw
(neighbourlist_2_closests), tol.solve = 1e-12, zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)
```

```
## 10  of  100
20  of  100
30  of  100
40  of  100
50  of  100
60  of  100
70  of  100
80  of  100
90  of  100
100  of  100
```

```
plot(autocorrelation_sar)
```



In this example you don't see much of an improvement (in fact the autocorrelation on short distances is increased), but if you were running the full data you would most likely see clearly that accounting for the closest neighbours improves the results. In general the measured autocorrelation in the neighbourhood model is expected to be smaller than in the GLM case above (general linear model) where we don't account for spatial auto-correlation.

Let's also check the model summary:

```
summary(sar_model)
```

```
##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = nb2listw(neighbourlist_2_closests), zero.policy = T,
##     tol.solve = 1e-12)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.4692486 -0.1902723  0.0094096  0.1956932  1.0928181
##
## Type: error
## Coefficients: (asymptotic standard errors)
##               Estimate Std. Error z value  Pr(>|z|)
## (Intercept) -0.0064187  0.0717876 -0.0894 0.9287544
## temperature  0.1649192  0.0432950  3.8092 0.0001394
##
## Lambda: 0.85053, LR test value: 1697.9, p-value: < 2.22e-16
## Asymptotic standard error: 0.008445
##     z-value: 100.71, p-value: < 2.22e-16
## Wald statistic: 10143, p-value: < 2.22e-16
##
## Log likelihood: -568.7366 for error model
## ML residual variance (sigma squared): 0.11511, (sigma: 0.33928)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1145.5, (AIC for lm: 2841.4)
```

We can see that the effect size of our predictor variable is much smaller than for the GLM model. This is expected because the predictor effect size in the GLM does not distinguish between the actual predictor and the effect size based on auto-correlation. The effect size of the SAR model on the other hand describes the true effect size of the predictor, not including biases caused by spatial-autocorrelation.

However we can improve this model further by testing different definitions of neighbourhood. Instead of calculating the distance to the n closest neighbours with the `knearneigh()` function, we can instead use the `dnearneigh()` function, which is based on a distance threshold and extracts all points within that distance. For example the following command extracts all points that are within a distance of 1500km of each point:

```
neighbours_1500km = dnearneigh(cbind(subsample$x,subsample$y), 0,1500, longlat
= T)
# This illustrates what we have done
plot(neighbours_1500km, cbind(subsample$x, subsample$y))
```

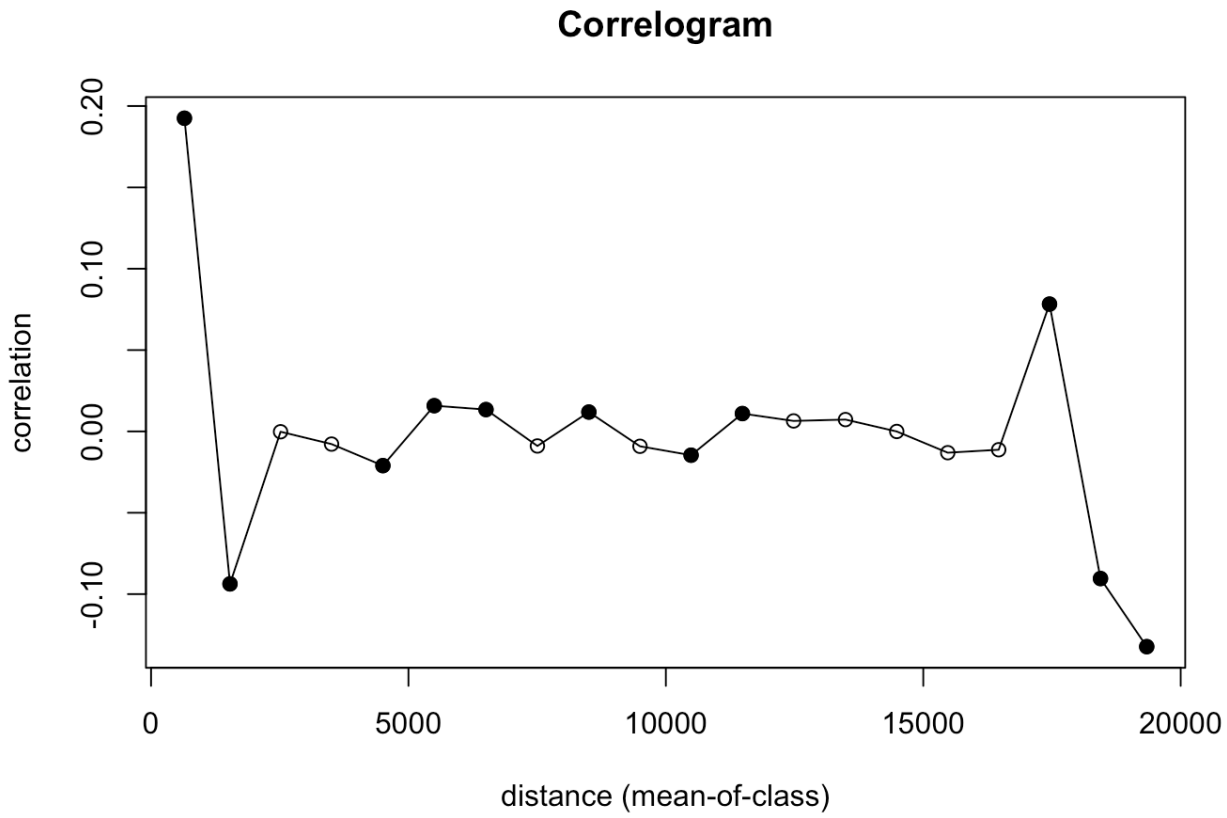
You see that the neighbourhoods look very different, since only points within 1500 km are connected. This also leads to some points having no neighbours at all, since they are too isolated. Also, note that the longer lines going across the whole map are a result of us using lon-lat input data (remember, the Earth is a globe).

Let's see how this neighbourhood definition performs compared to the n closests model from before. Note: The `zero.policy =T` argument in the SAR model and in the `nb2listw()` function is necessary to account for points without neighbours.

```
sar_model = errorsarlm(species_div~temperature, data=subsample, listw=nb2listw
(neighbours_1500km,zero.policy =T), tol.solve = 1e-12, zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)
```

```
## 10  of  100
20  of  100
30  of  100
40  of  100
50  of  100
60  of  100
70  of  100
80  of  100
90  of  100
100  of  100
```

```
plot(autocorrelation_sar)
```



The results don't look too different to before.

Let's also check the model summary:

```
summary(sar_model)
```

```
##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = nb2listw(neighbours_1500km, zero.policy = T), zero.policy = T,
##     tol.solve = 1e-12)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -2.585671 -0.213980  0.024525  0.291465  2.498851
##
## Type: error
## Regions with no neighbours included:
##  885
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept) -2.148077   0.425102 -5.0531 4.347e-07
## temperature  0.253407   0.051099  4.9591 7.082e-07
##
## Lambda: 0.98118, LR test value: 1335.6, p-value: < 2.22e-16
## Asymptotic standard error: 0.0062677
##     z-value: 156.55, p-value: < 2.22e-16
## Wald statistic: 24507, p-value: < 2.22e-16
##
## Log likelihood: -749.8798 for error model
## ML residual variance (sigma squared): 0.24249, (sigma: 0.49243)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1507.8, (AIC for lm: 2841.4)
```

Generally one should test a bunch of different configurations of the `knearneigh()` and `dnearneigh()` neighbourhoods, using different numbers of neighbours and different distances respectively. After testing different configurations the best model can be selected using a model selection criterion such as AIC. For that purpose we first define a function that runs through all our desired model configurations. Note: one can also try different values for the `style=` argument, common options are `B`, `C`, `U`, `S`, or `W` as used in the example below:

```
Neighborhood_generator=function(COOR) {
models<<-list(
  nb2listw(knn2nb(knearneigh(COOR,1, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,2, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,3, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,4, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,5, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,6, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,7, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,8, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,9, longlat = T)),style="W",zero.policy =T),
  nb2listw(knn2nb(knearneigh(COOR,10, longlat = T)),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="W",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="U",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,250, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,750, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1250, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,1500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,2500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3000, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,3500, longlat = T),style="S",zero.policy =T),
  nb2listw(dnearneigh(COOR, 0,4000, longlat = T),style="S",zero.policy =T)
)
}
```

Now we apply that list of models to our coordinates of the random subsample of our data:

```
neighbourhood_models = Neighborhood_generator(cbind(subsample$x,subsample$y))
```

Calculate the AICc score for each neighbourhood model using the `AICc()` function. This is a version of the regular AIC criterion that additionally corrects for small sample size:

```
library(wiqid)

AIC_LIST=numeric(length(neighbourhood_models))
for (i in 1:length(neighbourhood_models)) {
    AIC_LIST[i]=AICc(errorsarlm(species_div~temperature, data=subsample,listw=
neighbourhood_models[[i]], tol.solve = 1e-12, zero.policy =T))
}
```

Select the model with the lowest AIC score as the best model:

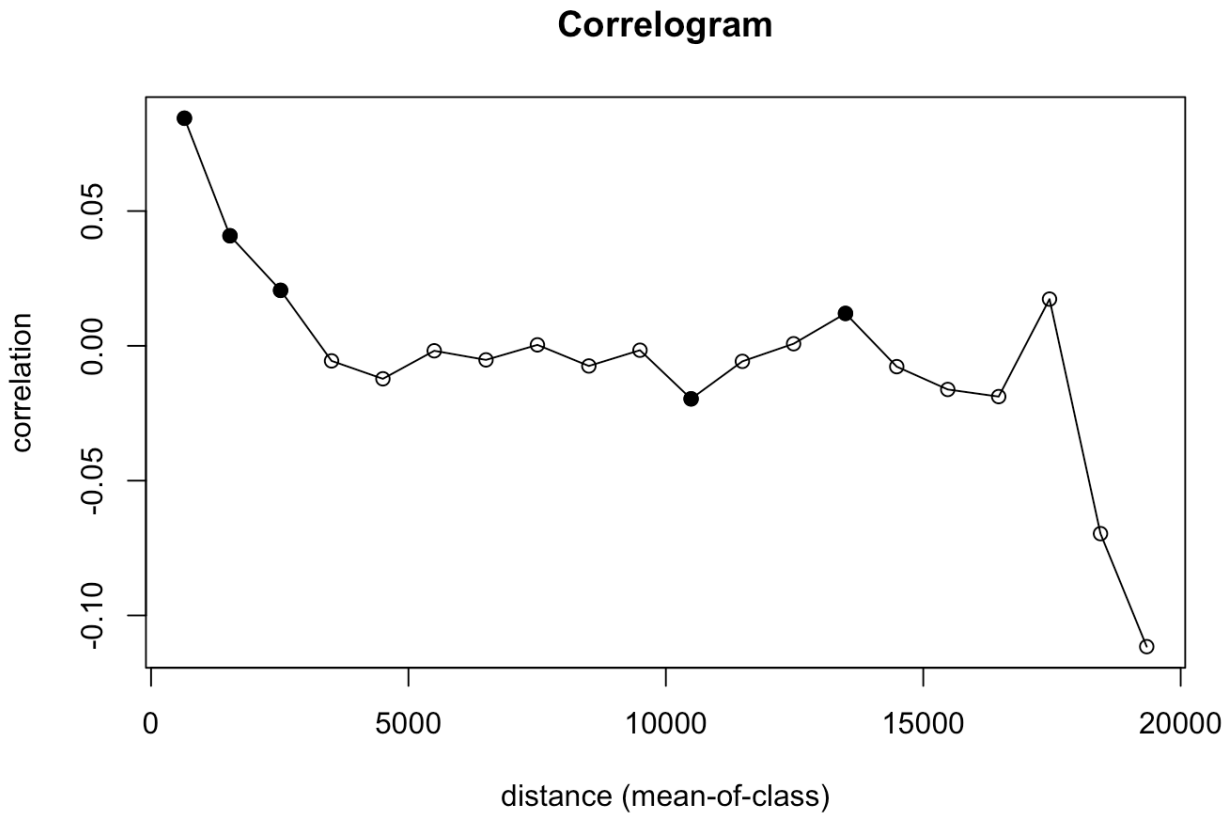
```
index_best_model = which(AIC_LIST==min(AIC_LIST))
best_neighbour_model = neighbourhood_models[index_best_model]
```

Plot the correlogram:

```
sar_model = errorsarlm(species_div~temperature, data=subsample, listw=best_nei
ghbour_model[[1]], tol.solve = 1e-12, zero.policy =T)
autocorrelation_sar = correlog(subsample$x, subsample$y, sar_model$residuals,
increment=1000, latlon=T, resamp=100)
```

```
## 10  of  100
20  of  100
30  of  100
40  of  100
50  of  100
60  of  100
70  of  100
80  of  100
90  of  100
100  of  100
```

```
plot(autocorrelation_sar)
```



This looks better, since the auto-correlation at small distances has almost disappeared (very small values). Again, this effect would probably be showing more cleraly if we were to run the whole dataset.

Let's check the model summary:

```
summary(sar_model)
```

```
##
## Call:errorsarlm(formula = species_div ~ temperature, data = subsample,
##     listw = best_neighbour_model[[1]], zero.policy = T, tol.solve = 1e-12)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.4799747 -0.1837494  0.0080152  0.1974077  1.9612638
##
## Type: error
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept) -0.021463   0.089848 -0.2389 0.8111941
## temperature  0.172396   0.046714  3.6904 0.0002239
##
## Lambda: 0.87941, LR test value: 1768.5, p-value: < 2.22e-16
## Asymptotic standard error: 0.0088698
##     z-value: 99.146, p-value: < 2.22e-16
## Wald statistic: 9830, p-value: < 2.22e-16
##
## Log likelihood: -533.4441 for error model
## ML residual variance (sigma squared): 0.11739, (sigma: 0.34263)
## Number of observations: 1000
## Number of parameters estimated: 4
## AIC: 1074.9, (AIC for lm: 2841.4)
```

Determine model fit

Besides using the `summary()` function to check the size of the effect of the predictor and the significance of it, you can get a measure of model fit by calculating the pseudo R-square value.

```
# This gives a pseudo R square for the glm
cor(predict(glm_model), subsample$species_div)^2
```

```
## [1] 0.03922745
```

```
# This gives a pseudo R square for the SAR but for both the predictors and the
neighborhood part
cor(predict(sar_model),  subsample$species_div)^2
```

```
## [1] 0.895755
```

The R-square value for the SAR model is much higher, which is expected because it contains the spatial information as well and we saw that the data has very strong autocorrelation. Therefore this value does not tell us much about the predictive power of our actual predictor variable, but only of the whole model. To get an idea of the R-square for only the predictor we can create another model solely containing the neighborhood:

```
sar_model_NULL = errorsarlm(species_div~1, data=subsample, listw=best_neighbou
r_model[[1]], tol.solve = 1e-12, zero.policy =T)
cor(predict(sar_model_NULL), subsample$species_div)^2
```

```
## [1] 0.8951924
```

Further steps (not included in this tutorial):

- Run the selected best model on the full dataset (instead of the subsample we used above)
- Add multiple predictors (at least 2) in one joined analysis. The general syntax to include multiple predictors into the model is e.g. `glm(species_div~predictor1+predictor2,data=subsample)` , or if you want to include interactions between predictors `glm(species_div~predictor1*predictor2,data=subsample)` (equivalent syntax for the SAR model).