SpatialRDD: get started

Alexander Lehner

2019-05-26

Contents

Within the last years, the use of spatial version of Regression Discontinuity Designs has increased tremendously in popularity in the social sciences, most importantly Economics and Political Science. `SpatialRDD` is the first (geo-)statistical package of its kind that unifies the geographic tasks that are needed for Spatial Regression Discontinuity Designs with all potential parametric and non-parametric estimation techniques that have been put forward (see Lehner2019). Geographic objects are treated as simple features throughout, making heavy use of the novel `sf` package by Edzer Pebesma which revolutionised spatial data analysis in **R** and is bound to supersede the older and less versatile `sp` package.
`SpatialRDD` facilitates analysis inter alia because it contains all necessary functions to automatise otherwise very tedious tasks that are tipically carried out in GUIs such as QGIS or ArcGIS. Due to the fact that these GIS interfaces are not able to carry out appropriate statistical analysis, the researcher is tipically forced to migrate the obtained results to statistical software. This makes reproducibility difficult and most importantly is a very inefficient workflow.
`SpatialRDD` unifies everything in one language and e.g. has the necessary functions to check for different bandwiths, shift placebo boundaries, do all necessary distance calculations, assign treated/non-treated dummies, and flexibly assign border segment fixed effects while keeping the units of observations at their proper position in space and allowing the researcher to visualise every intermediate step with mapplots. For the latter we will mostly rely on the flexible and computationally very efficient `tmap` package, while also `ggplot2` is used at times.
For the purpose of illustration, this vignette uses simulated data on real boundaries/polygons and guides the user through every necessary step in order to carry out a Geographic RDD estimation. At the appropriate points we will also make remarks on technical caveats and issues that have been pointed out in the literature and give suggestions to improve these designs.
The workhorse functions of `SpatialRDD` in a nutshell are:

- `assign_treated()`
- `border_segment()`
- `discretise_border()`
- `SpatialRD()`
- `plotspatialrd()`

- `placebo_border()`
- `cutoff2polygons()`

Some words of caution

all the projections system importance plus the story mit GADM is dangerous (visualise with leaflet odersowas auf openstreetmap)

Setup and Propaedeutics

Throughout the vignette we will use the geographic boundaries on Goa, India, from Lehner (2019). The data, included in the package, contains

- a line called `cut_off.sf` which resembles the spatial discontinuity
- a polygon that defines the "treated" area
- a polygon that defines the full study area (which is going to be useful as this defines the bounding box)

```
library(SpatialRDD)
library(sf)
data(Goa_GIS)
```

These come in the EPSG 32643 projection system, which is a "localised" UTM coordinate system. These are generally prefereable for exercises like a Spatial RDD, as they are more precise and also allow us to work in metres (the "classic" longitude/latitude CRS, EPSG 4326, works in degrees). If your study area is small, you should think about reprojecting your data into the CRS of the according UTM zone (simply use `st_transform()`).

All the spatial objects are of class `sf` from the sf package. This means they are just a `data.frame` with a special column that contains a geometry for each row. The big advantage is, no matter if you prefer base R, `dplyr`, or any other way to handle and wrangle your data, the `sf` object can be treated just like a standard `data.frame`. The one single step that transforms these spatial objects back to a standard `data.frame` is just dropping the geometry column with

```
st_geometry(any.sf.object) <- NULL
```

or alternatively

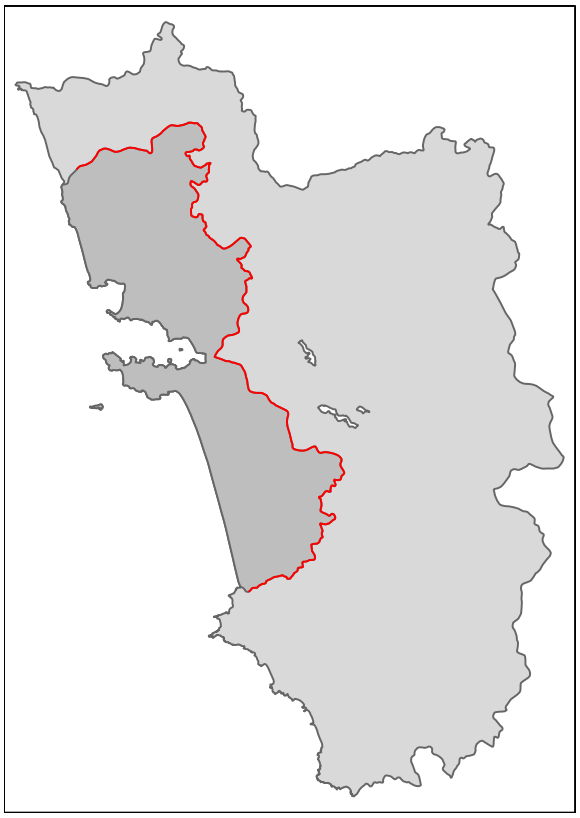
```
st_set_geometry(any.sf.object, NULL)
```

If you import geospatial data in a different format, say the common shapefile (`*.shp`) - which is NOT preferrable see why here, or as a geopackage, it is fairly straightforward to convert it:

```
mydata.sf <- st_as_sf(loaded_file, coords = c("longitude", "latitude"), crs = projcrs)
# just the EPSG or a proj4string
```

Inspecting the Study Area & simulating Data

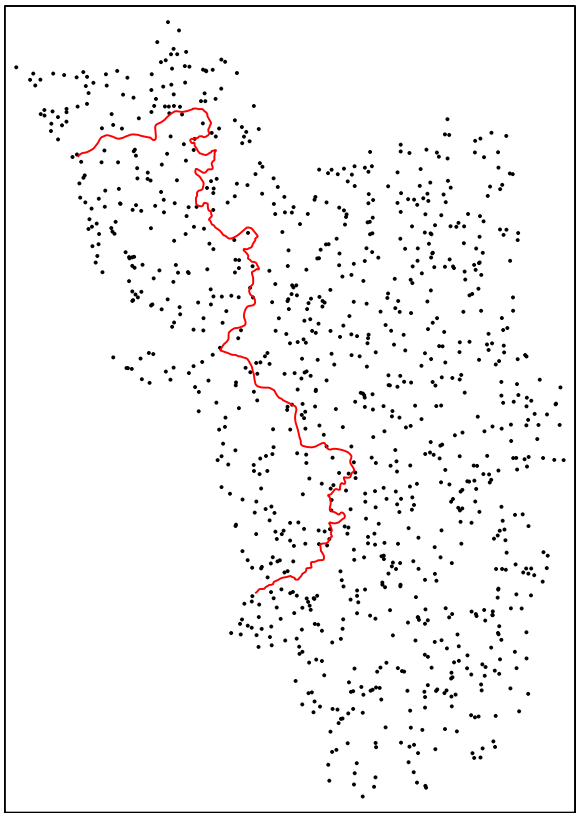
```
library(tmap)
tm_shape(polygon_full.sf) + tm_polygons() +
  tm_shape(polygon_treated.sf) + tm_polygons(col = "grey") +
  tm_shape(cut_off.sf) + tm_lines(col = "red")
```



Above we see the simple map, visualising the "treated polygon" in a darker grey, and the `tmap` syntax that produced it.

Let's simulate some random points within the polygon that describes the full study area:

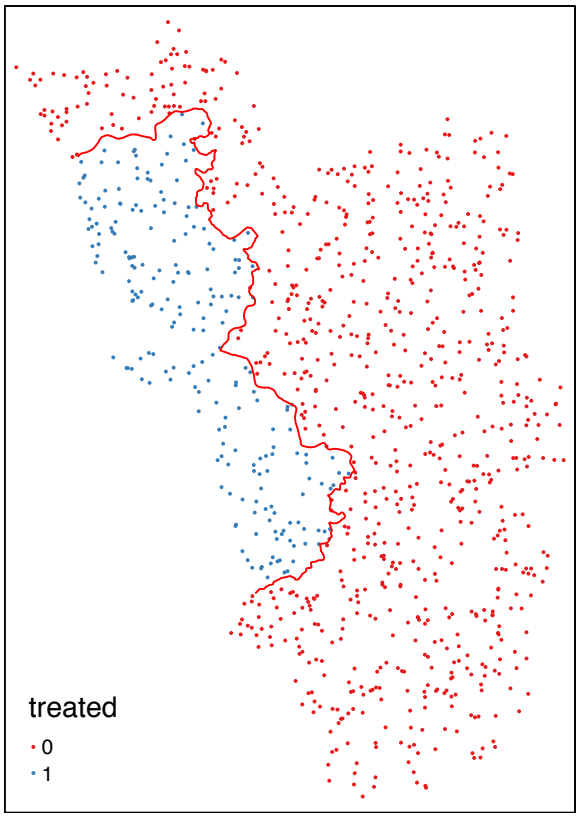
```
set.seed(1088) # set a seed to make the results replicable
points_samp.sf <- sf::st_sample(polygon_full.sf, 1000)
points_samp.sf <- sf::st_sf(points_samp.sf) # make it an sf object bc st_sample just created the geomet
points_samp.sf$id <- 1:nrow(points_samp.sf) # add a unique ID to each observation
# visualise results together with the line that represents our RDD cut-off
tm_shape(points_samp.sf) + tm_dots() + tm_shape(cut_off.sf) + tm_lines(col = "red")
```



Assign Treatment

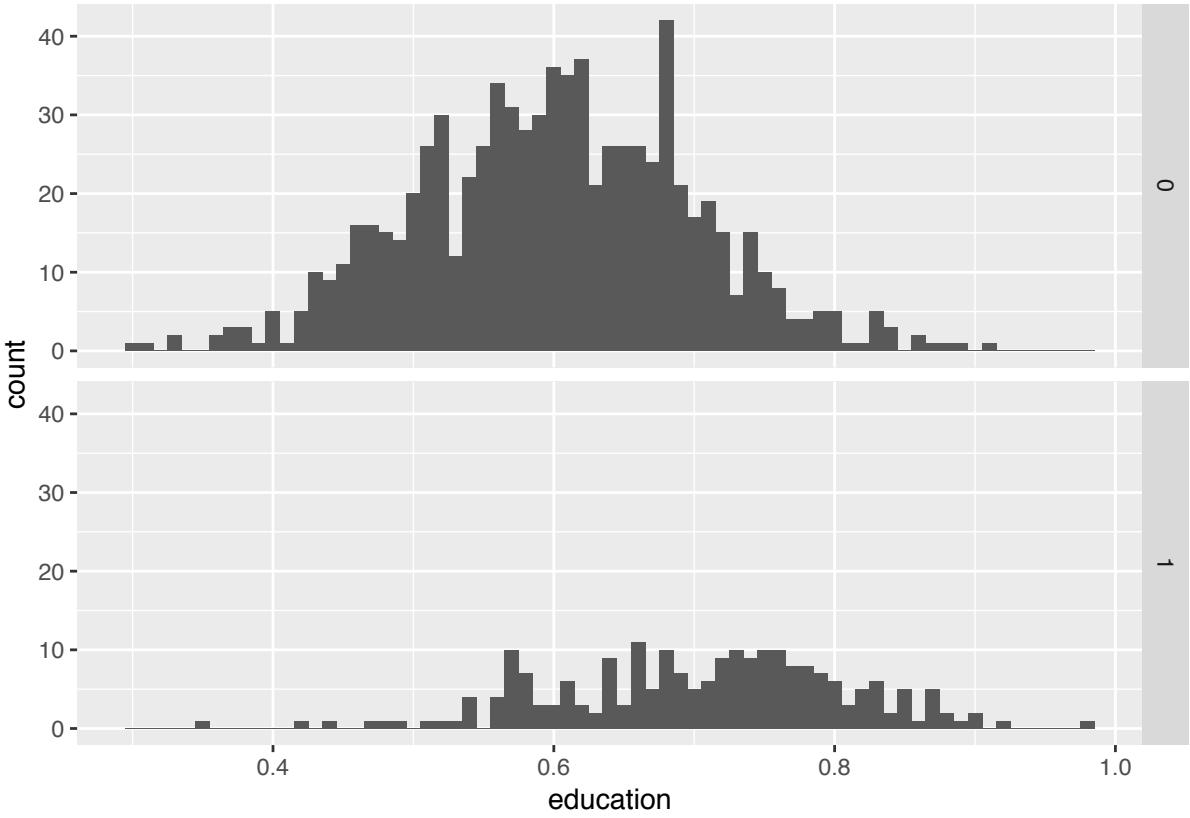
Now we use the first function of the `SpatialRDD` package which is called `assign_treated()` which in essence does a spatial intersection and returns a column vector that contains `0` or `1` depending on whether the village is inside or outside the treatment area. Thus we just add it as a new column to the points object. The function requires the name of the points object, the name of the polygon that defines the treated area, and the id that uniquely identifies each observation in the points object:

```
points_samp.sf$treated <- assign_treated(points_samp.sf, polygon_treated.sf, id = "id")
tm_shape(points_samp.sf) + tm_dots("treated", palette = "Set1") + tm_shape(cut_off.sf) + tm_lines(col =
```



As a next step we add an outcome of interest that we are going to use as dependent variable in our Spatial Regression Discontinuity Design. Let's call this variable `education`, measuring the literacy rate that ranges from 0 to 1 (0%, meaning everyone illiterate to 100%, meaning everyone in the population can read and write). We assume that the units, call them villages, in the "treated" polygon have on average a higher literacy rate because they received some sort of "treatment". Let's just assume aliens dropped (better) schools in all of these villages, but NOT in any of the outside villages, and everything else is constant and identical across the two territories.

```
# first we define a variable for the number of "treated" and control which makes the code more readable
NTr <- length(points_samp.sf$id[points_samp.sf$treated == 1])
NCo <- length(points_samp.sf$id[points_samp.sf$treated == 0])
# the treated areas get a 10 percentage point higher literacy rate
points_samp.sf$education[points_samp.sf$treated == 1] <- 0.7
points_samp.sf$education[points_samp.sf$treated == 0] <- 0.6
# and we add some noise, otherwise we would obtain regression coeffictions with no standard errors
# we draw from a normal with mean 0 and a standard devation of 0.1
points_samp.sf$education[points_samp.sf$treated == 1] <- rnorm(NTr, mean = 0, sd = .1) +
  points_samp.sf$education[points_samp.sf$treated == 1]
points_samp.sf$education[points_samp.sf$treated == 0] <- rnorm(NCo, mean = 0, sd = .1) +
  points_samp.sf$education[points_samp.sf$treated == 0]
# visualisation split up by groups
library(ggplot2)
ggplot(points_samp.sf, aes(x = education)) + geom_histogram(binwidth = .01) + facet_grid(treated ~ .)
```



From the above histograms we can see that we were successful in creating different group means. This is also confirmed by the simple univariate regression of $Y_i = \alpha + \beta T_i + \varepsilon_i$:

```
summary(lm(education ~ treated, data = points_samp.sf))
#>
#> Call:
#> lm(formula = education ~ treated, data = points_samp.sf)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -0.36005 -0.06767  0.00299  0.07162  0.30610
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 0.600247   0.003536  169.74   <2e-16 ***
#> treated1    0.104941   0.007616   13.78   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.09959 on 1009 degrees of freedom
#> Multiple R-squared:  0.1584, Adjusted R-squared:  0.1575
#> F-statistic: 189.9 on 1 and 1009 DF,  p-value: < 2.2e-16
```

where the intercept tells us that the average in the non-treated areas is 0.6 and treated villages have on average 0.1 more education (10 percentage points).

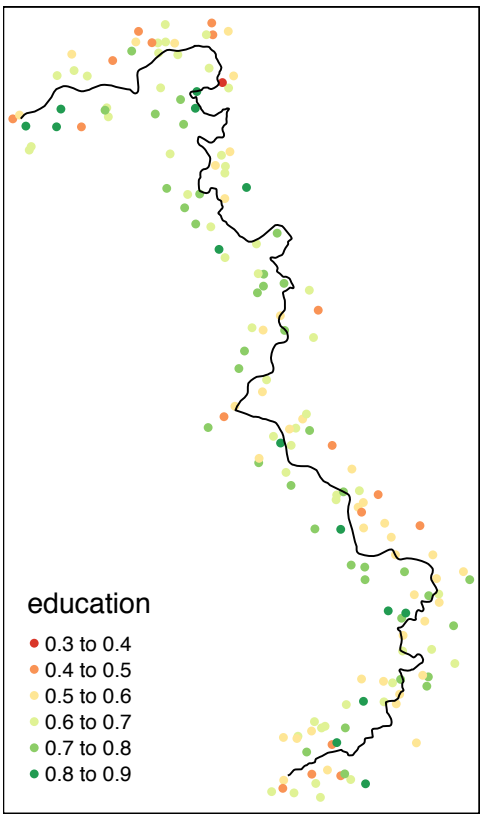
Distance to Cut-off

The next essential step before we start to do proper Spatial RDD analysis, is to determine how far each of these points is away from the cut-off. Here we just make use of a function from `sf` called `st_distance()` that returns a vector with units (that we have to convert to real numbers by `as.numeric()`):

```
points_samp.sf$dist2cutoff <- as.numeric(sf::st_distance(points_samp.sf, cut_off.sf))
```

This allows us now to investigate villages only within a certain range, say 3 kilometres, around our "discontinuity":

```
tm_shape(points_samp.sf[points_samp.sf$dist2cutoff < 3000, ]) + tm_dots("education", palette = "RdYlGn"
  tm_shape(cut_off.sf) + tm_lines()
```



And to run the univariate regression from above also just within a bandwith (this specification is already starting to resemble the RDD idea of Dell 2010). As we know the exact data generating process (no "spatial gradiant" but a rather uniform assignment), it is obvious to us that this of course leaves the point estimate essentially unchanged:

```
summary(lm(education ~ treated, data = points_samp.sf[points_samp.sf$dist2cutoff < 3000, ]))
#>
#> Call:
#> lm(formula = education ~ treated, data = points_samp.sf[points_samp.sf$dist2cutoff <
#>     3000, ])
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.259185 -0.058693  0.003819  0.060435  0.291483
#>
```

```
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  0.59202    0.01048  56.496  < 2e-16 ***
#> treated1     0.11130    0.01505   7.395 7.34e-12 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.09604 on 161 degrees of freedom
#> Multiple R-squared:  0.2535, Adjusted R-squared:  0.2489
#> F-statistic: 54.68 on 1 and 161 DF,  p-value: 7.343e-12
```

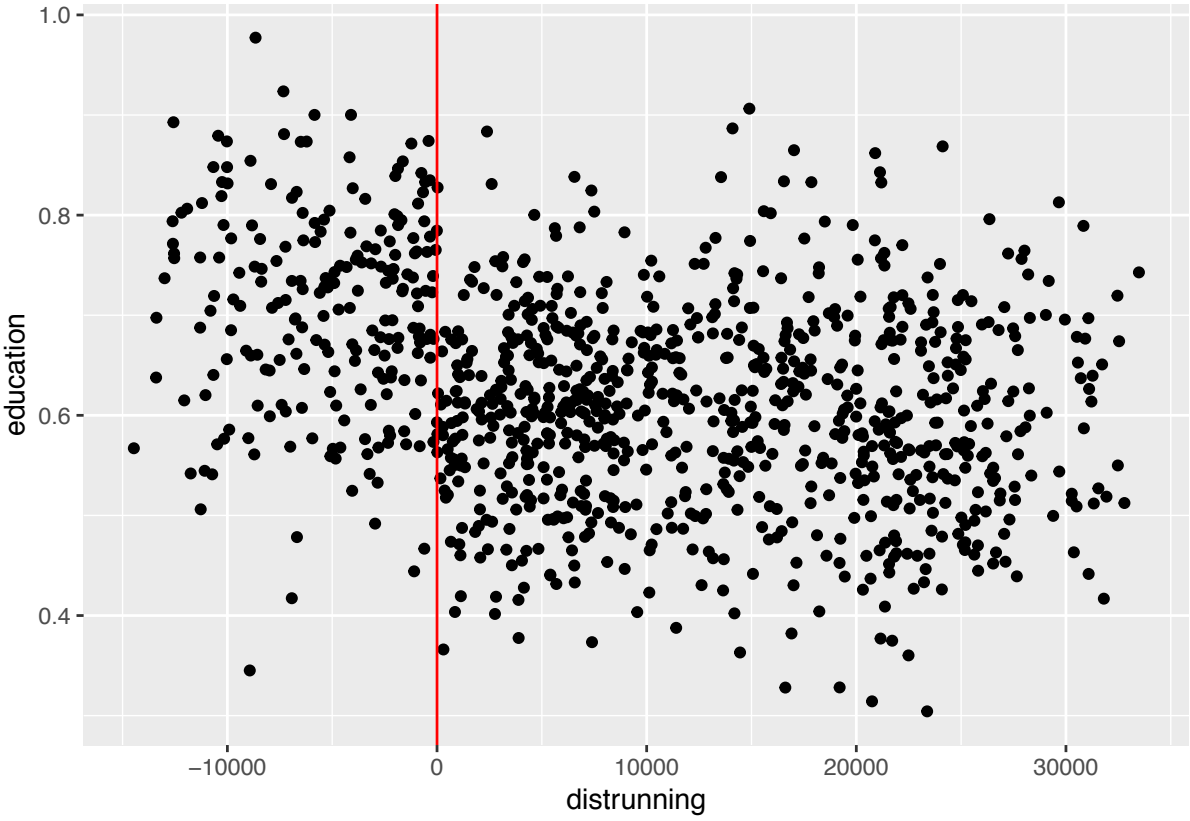
Spatial RDD estimation

Now we go step by step through all potential (parametric and non-parametric) ways in which one could obtain point estimates for Spatial RDD's (see e.g. Lehner2019 for details).

Naive Distance

For the "naive" estimation (KeeleTitiunik2015), meaning that the spatial dimension is essentially disregarded, we first define a variable `distrunning` that makes the distances in the treated areas negative so that our 2-dimensional cut-off is then at 0.

```
points_samp.sf$distrunning <- points_samp.sf$dist2cutoff
points_samp.sf$distrunning[points_samp.sf$treated == 1] <- -1 * points_samp.sf$distrunning[points_samp.s
ggplot(data = points_samp.sf, aes(x = distrunning, y = education)) + geom_point() + geom_vline(xintercep
```

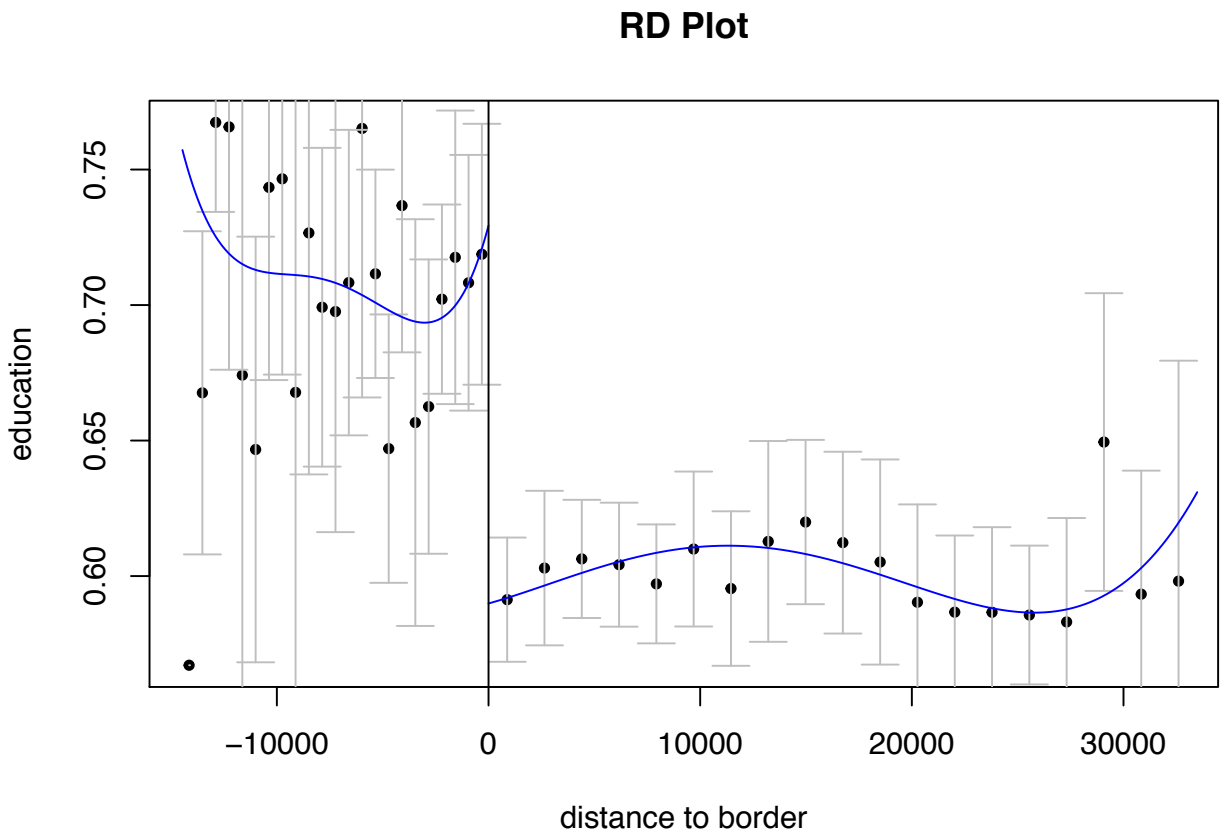


The point estimate of the "classic" non-parametric local linear regression, carried out with the `rdrobust` package, then looks like this:

```
library(rdrobust)
summary(rdrobust(points_samp.sf$education, points_samp.sf$distrunning, c = 0))
#> Call: rdrobust
#>
#> Number of Obs.                 1011
#> BW type                       mserd
#> Kernel                   Triangular
#> VCE method                       NN
#>
#> Number of Obs.                 218         793
#> Eff. Number of Obs.            101         126
#> Order est. (p)                   1           1
#> Order bias  (p)                  2           2
#> BW est. (h)               4178.397    4178.397
#> BW bias (b)               6736.288    6736.288
#> rho (h/b)                    0.620       0.620
#>
#> ================================================================================
#>         Method     Coef. Std. Err.         z     P>|z|      [ 95% C.I. ]
#> ================================================================================
#>   Conventional    -0.142      0.027    -5.362     0.000    [-0.194 , -0.090]
#>         Robust         -          -    -4.756     0.000    [-0.211 , -0.088]
#> ================================================================================
```

and the according visualisation with data driven bin-width selection:

```
rdplot(points_samp.sf$education, points_samp.sf$distrunning, c = 0, ci = 95,
       kernel = "triangular", y.label = "education", x.label = "distance to border")
#> Warning in arrows(rdplot_mean_bin, rdplot_cil_bin, rdplot_mean_bin,
#> rdplot_cir_bin, : zero-length arrow is of indeterminate angle and so
#> skipped
```



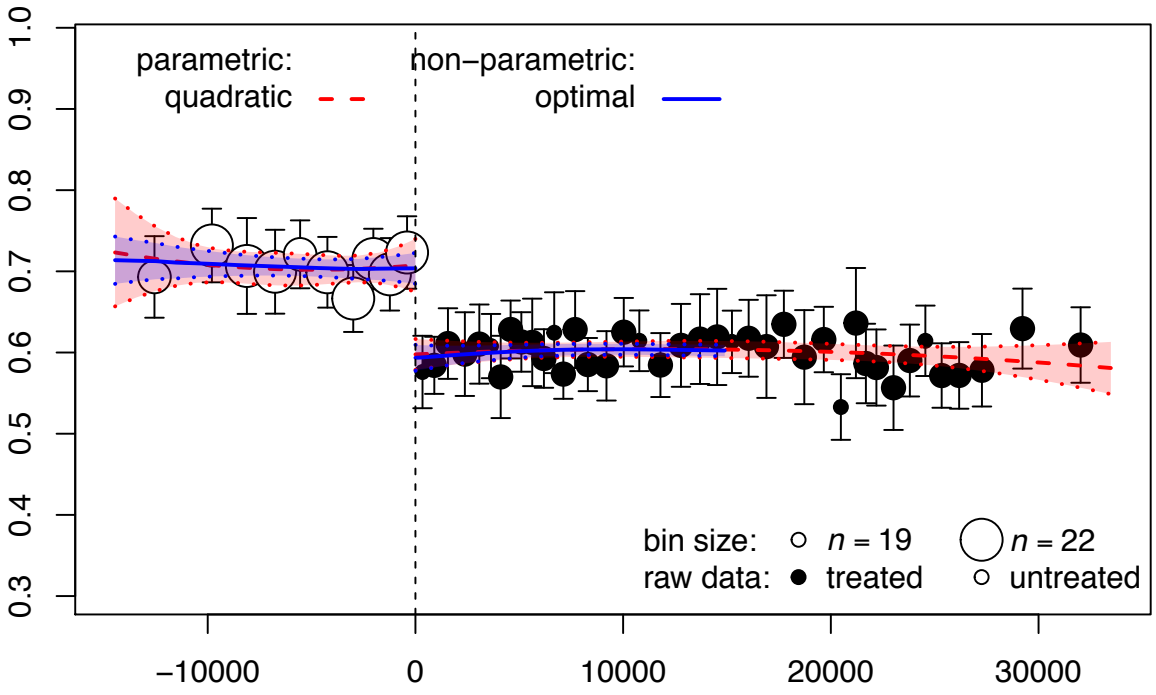
For RDD estimation in **R** in general there are currently three packages flying around: `RDD`, `rddtools`, and `rddapp` (building on the `RDD`); whereby the latter seems to be the most up-to-date and comprehensive one (as it draws on previous work that others did).
`rddapp` estimates various specifications (does not do robust inference though)

```
library(rddapp)
summary(rd_est(education ~ distrunning, data = points_samp.sf, t.design = "g"))
#>
#> Call:
#> rd_est(formula = education ~ distrunning, data = points_samp.sf,
#>     t.design = "g")
#>
#> Type:
#> sharp
#>
#> Estimates:
#>             Bandwidth  Observations  Estimate  Std. Error  lower.CL
#> Linear         NA       1011         -0.09772  0.01297     -0.1231
#> Quadratic      NA       1011         -0.10974  0.01881     -0.1466
#> Cubic          NA       1011         -0.14279  0.02397     -0.1898
#> Opt         15710        679         -0.11018  0.01499     -0.1396
```

```
#> Half-Opt      7855       428          -0.12427  0.01943     -0.1624
#> Double-Opt   31419      1002          -0.10086  0.01322     -0.1268
#>              upper.CL  z value  Pr(>|z|)
#> Linear       -0.07229  -7.533   4.967e-14  ***
#> Quadratic    -0.07288  -5.835   5.393e-09  ***
#> Cubic        -0.09582  -5.958   2.551e-09  ***
#> Opt          -0.08080  -7.348   2.007e-13  ***
#> Half-Opt     -0.08618  -6.395   1.606e-10  ***
#> Double-Opt   -0.07495  -7.628   2.387e-14  ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Confidence interval used:  0.95
```

And it gives several possibilities of visualising classic RDDs. Here we arbitrarily pick one parametric and one non-parametric estimate, including confidence intervals, and manually chosen binsizes:

```
plot(rd_est(education ~ distrunning, data = points_samp.sf, t.design = "g"), fit_line = c("quadratic",
```

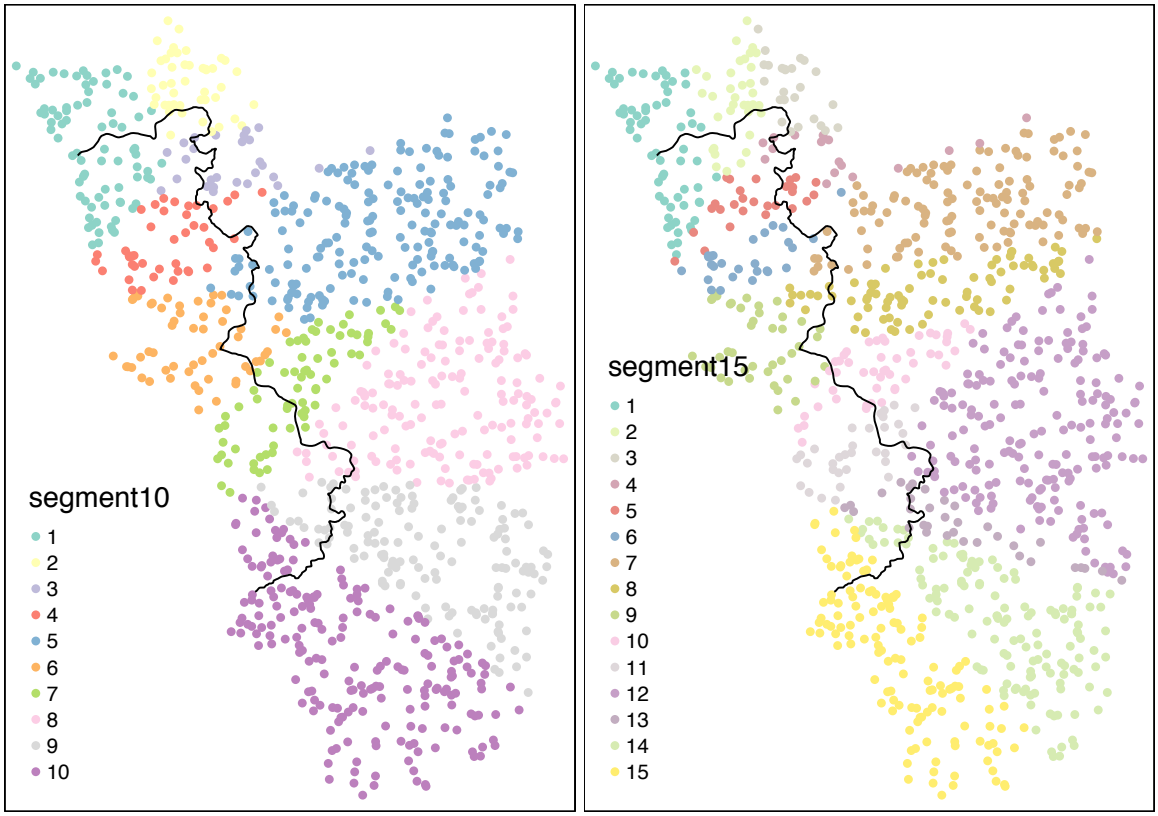


Polynomial Specifications

This method, popularised by Dell2010 in her JMP on the Peruvian Mining Mita, examines only observations within a certain distance around the border by using a (semi-)parametric approach. From the point of view of which additional "spatial technicalities" are needed, it essentially only boils down to the introduction of border segments. These are then used to apply a "within estimator" to allow for different slopes for each of those segment categories in order to, inter alia, alleviate the omitted variable problem. As an alternative to this fixed-effects approach we might as well throw a set of dummies for each of the segments in the regression. The regression coefficient of interest then gives a weighted average over all segments. On top of that we might also be interested in the coefficient of each segment to infer something about potential heterogeneity alongside our regression discontinuity.
The (computationally a bit demanding) function `border_segment()` only needs the points layer and the

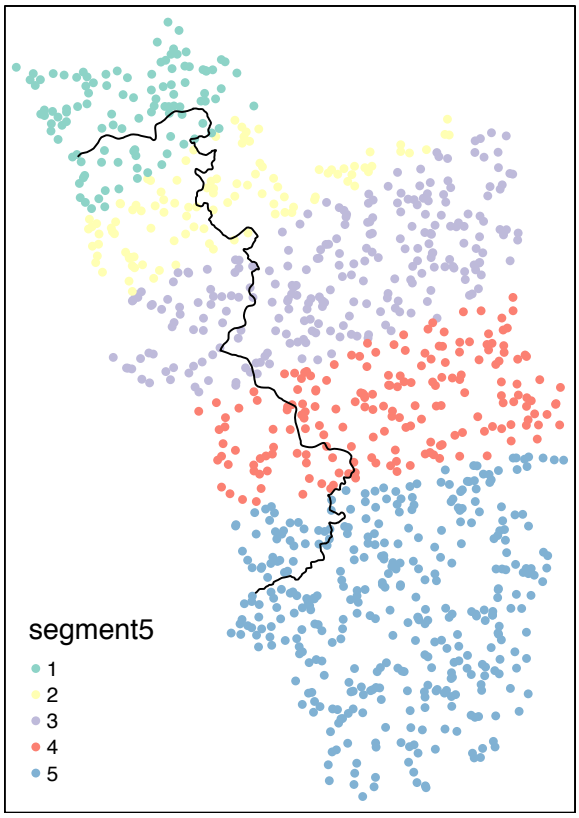
cut-off as input (preferrably as line, but also an input in the form of boundarypoint works). The last parameter of the function let's us determine how many segments we want. As with the `assign_treated()` function, the output is a vector of factors.

```
points_samp.sf$segment10 <- border_segment(points_samp.sf, cut_off.sf, 10)
#> Starting to create 10 border segments with an approximate length of 13 kilometres each.
points_samp.sf$segment15 <- border_segment(points_samp.sf, cut_off.sf, 15)
#> Starting to create 15 border segments with an approximate length of 9 kilometres each.
tm_shape(points_samp.sf) + tm_dots("segment10", size = 0.1) + tm_shape(cut_off.sf) + tm_lines()
tm_shape(points_samp.sf) + tm_dots("segment15", size = 0.1) + tm_shape(cut_off.sf) + tm_lines()
```



It is worth noting that the researcher has to pay attention to how the fixed effects are assigned. It could, e.g. due to odd bendings of the cut-off, be the case that for some segment only one side actually gets assigned points or the like. These situations are undesireable for estimation and a visualisation of how the segments are distributed across space is paramount.
The `border_segment()` already gives the researcher a feeling for how meaningful the choice for the number of segments was. In the above example we have a segment for every 13 kilometres, which seems not too unreasonable.
In the following example we choose less borderpoints, leading to more observations on each side of the border for every segment and thus to more meaningful point estimates:

```
points_samp.sf$segment5 <- border_segment(points_samp.sf, cut_off.sf, 5)
#> Starting to create 5 border segments with an approximate length of 26 kilometres each.
tm_shape(points_samp.sf) + tm_dots("segment5", size = 0.1) + tm_shape(cut_off.sf) + tm_lines()
```



Simple OLS estimates, using the segments that we just obtained as categories for our fixed effects, show these differences (which in our simulated case of course are negligible):

```
library(lfe)
#> Warning: package 'lfe' was built under R version 3.5.2
#> Loading required package: Matrix
summary(felm(education ~ treated | factor(segment10) | 0 | 0, data = points_samp.sf[points_samp.sf$dist
#>
#> Call:
#>    felm(formula = education ~ treated | factor(segment10) | 0 |      0, data = points_samp.sf[points
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.247552 -0.056219  0.001652  0.067336  0.256214
#>
#> Coefficients:
#>          Estimate Robust s.e t value Pr(>|t|)
#> treated1   0.1043     0.0164   6.359 2.26e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.0957 on 152 degrees of freedom
#> Multiple R-squared(full model): 0.3002   Adjusted R-squared: 0.2542
#> Multiple R-squared(proj model): 0.2231   Adjusted R-squared: 0.172
#> F-statistic(full model, *iid*): 6.52 on 10 and 152 DF, p-value: 2.404e-08
#> F-statistic(proj model): 40.43 on 1 and 152 DF, p-value: 2.258e-09

summary(felm(education ~ treated | factor(segment5) | 0 | 0, data = points_samp.sf[points_samp.sf$dist2
```

```
#>
#> Call:
#>    felm(formula = education ~ treated | factor(segment5) | 0 | 0,      data = points_samp.sf[points_
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.252130 -0.061236  0.001951  0.057814  0.255506
#>
#> Coefficients:
#>           Estimate Robust s.e t value Pr(>|t|)
#> treated1   0.1084     0.0153   7.087 4.36e-11 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.0962 on 157 degrees of freedom
#> Multiple R-squared(full model): 0.2697   Adjusted R-squared: 0.2464
#> Multiple R-squared(proj model): 0.2425   Adjusted R-squared: 0.2184
#> F-statistic(full model, *iid*):11.59 on 5 and 157 DF, p-value: 1.527e-09
#> F-statistic(proj model): 50.23 on 1 and 157 DF, p-value: 4.36e-11
```

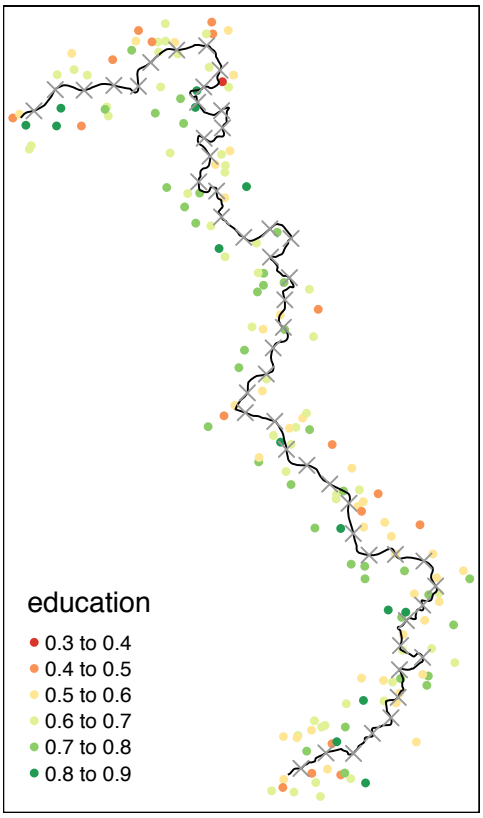
We obtain a point estimate that is (unsurprisingly, as we have a data generating process that is very uniform across space) very similar to the one we obtained from the simple OLS regression from the beginning. As compared to the "classic RD" point estimate that we obtained from the non-parametric local linear regression from the `rdrobust` package, the point estimate from our fixed effects regression is a bit more conservative. But from eyeballing we can determine that the average effect lies somewhere around 0.1, meaning that the literacy rate is 10 percentage points higher in the treated areas. Exactly the way we designed our simulated data.
In the "full" polynomial specification by Dell2010 we would be required to also control for the position in space via a flexible polynomial function in longitude and latitude.

GRD

Finally we move towards a fully fledged Geographic Regression Discontinuity (GRD) design (KeeleTitiunik2015). The function `SpatialRD()` incorporates the RD estimation with two running variables, but also allows to carry out the estimation on each boundarypoint ("GRDDseries") with just one line of code. This allows us to visualise the treatment effect at multiple points of the cut-off and thus infer something about the potential heterogeneity of the effect. Or, most importantly, to assess the robustness of the GRD itself.
A future version of `SpatialRDD` will also incorporate the "Optimized RDD" approach by ImbensWager2019.

First of all we have to cut the border into equally spaced segments. For each of these segments, or rather boundarypoints, we will later obtain a point estimate. The `discretise_border()` function just requires the sf object that represent the cut-off (polyline preferred but also points possible) and the number of desired boundarypoints:

```
borderpoints.sf <- discretise_border(cutoff = cut_off.sf, n = 50)
#> Starting to create 50 borderpoints. Approximately every 3 kilometres we can run an estimation then.
tm_shape(points_samp.sf[points_samp.sf$dist2cutoff < 3000, ]) + tm_dots("education", palette = "RdYlGn"
  tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(borderpoints.sf) + tm_symbols(shape = 4, size = .3)
```



for plotting just a results table, it would be preferrable to choose just a `data.frame` as output (spatial.object = F).

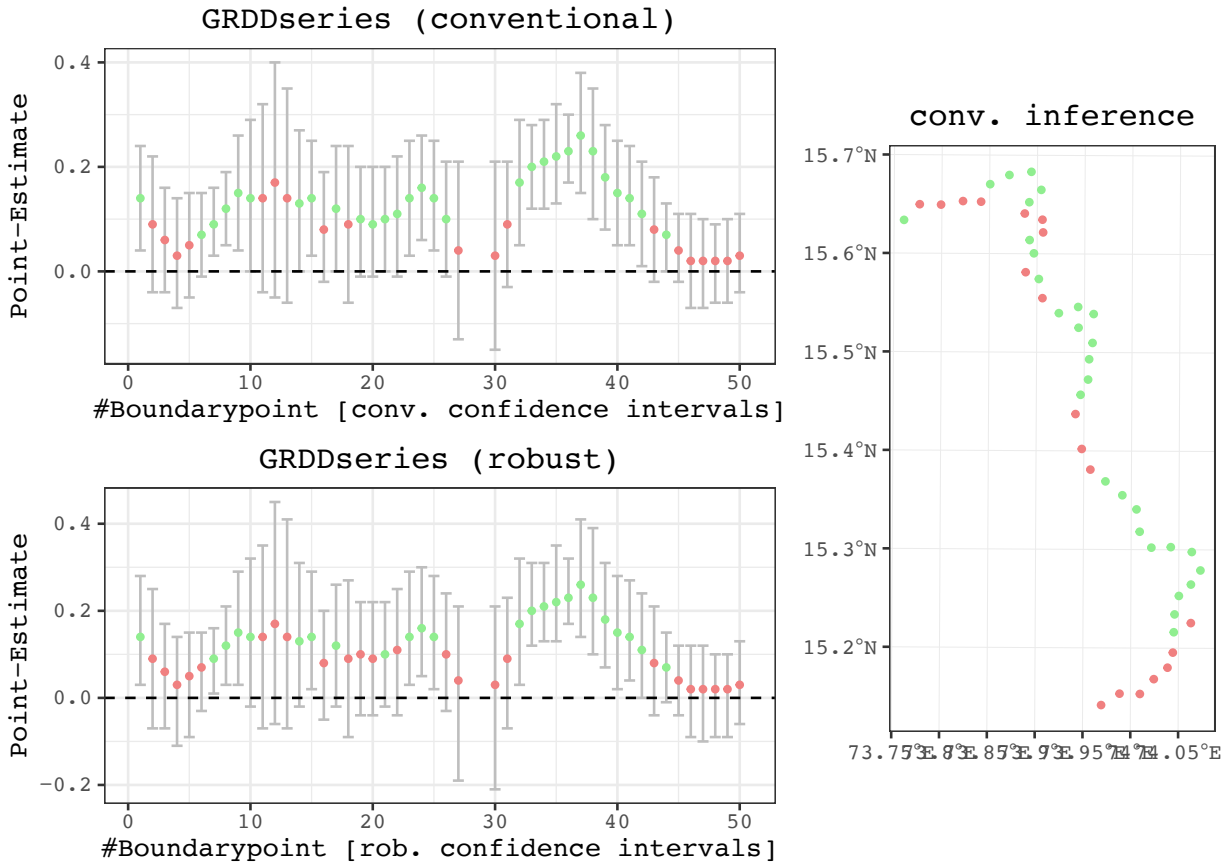
```
results <- SpatialRD(y = "education", data = points_samp.sf, cutoff.points = borderpoints.sf, treated =
#> We have 1011 observations of which 218 are treated observations.
#> We are iterating over 50 Boundarypoints.
#> The dependent variable is education .
knitr::kable(results)
```

Point	Estimate	pvalC	pvalR	Ntr	Nco	bw	CI_Conv_l	CI_Conv_u	CI_Rob_l	CI_Rob_u	McCrary
1	0.14	0.01	0.01	52	54	13.5	0.04	0.24	0.03	0.28	NA
2	0.09	0.15	0.26	74	59	15.3	-0.04	0.22	-0.07	0.25	NA
3	0.06	0.23	0.45	104	84	19.4	-0.04	0.16	-0.07	0.17	NA
4	0.03	0.52	0.81	97	60	15.0	-0.07	0.14	-0.11	0.14	NA
5	0.05	0.34	0.61	102	70	16.5	-0.05	0.15	-0.09	0.15	NA
6	0.07	0.08	0.21	112	80	19.7	-0.01	0.15	-0.03	0.15	NA
7	0.09	0.01	0.03	123	87	21.6	0.03	0.16	0.01	0.16	NA
8	0.12	0.00	0.01	122	73	20.6	0.05	0.19	0.03	0.21	NA
9	0.15	0.01	0.02	107	60	18.3	0.04	0.26	0.03	0.29	NA
10	0.14	0.06	0.09	86	50	15.4	0.00	0.29	-0.02	0.32	NA
11	0.14	0.14	0.18	78	47	14.0	-0.04	0.32	-0.07	0.35	NA
12	0.17	0.13	0.13	54	28	10.8	-0.05	0.40	-0.06	0.45	NA
13	0.14	0.17	0.17	41	29	9.9	-0.06	0.35	-0.07	0.41	NA
14	0.13	0.06	0.09	52	45	12.2	0.00	0.27	-0.02	0.31	NA
15	0.14	0.01	0.03	50	51	12.1	0.03	0.25	0.02	0.29	NA
16	0.08	0.12	0.24	75	113	16.2	-0.02	0.19	-0.05	0.20	NA
17	0.12	0.05	0.09	45	77	13.1	0.00	0.24	-0.02	0.26	NA

Point	Estimate	pvalC	pvalR	Ntr	Nco	bw	CI_Conv_l	CI_Conv_u	CI_Rob_l	CI_Rob_u	McCrary
18	0.09	0.22	0.32	56	100	14.4	-0.06	0.24	-0.09	0.27	NA
19	0.10	0.07	0.19	132	122	18.4	-0.01	0.20	-0.04	0.22	NA
20	0.09	0.08	0.16	96	76	15.3	-0.01	0.20	-0.04	0.22	NA
21	0.10	0.05	0.09	81	38	13.0	0.00	0.20	-0.02	0.22	NA
22	0.11	0.08	0.15	79	55	13.5	-0.01	0.22	-0.04	0.25	NA
23	0.14	0.01	0.02	87	43	12.9	0.03	0.25	0.03	0.29	NA
24	0.16	0.00	0.01	74	48	12.6	0.06	0.26	0.05	0.30	NA
25	0.14	0.01	0.02	84	58	13.4	0.04	0.25	0.02	0.28	NA
26	0.10	0.06	0.13	61	59	12.6	-0.01	0.21	-0.03	0.24	NA
27	0.04	0.68	0.89	29	32	8.6	-0.13	0.21	-0.19	0.21	NA
30	0.03	0.77	0.99	25	30	8.6	-0.15	0.21	-0.21	0.21	NA
31	0.09	0.14	0.28	55	54	12.0	-0.03	0.21	-0.07	0.23	NA
32	0.17	0.01	0.02	48	41	10.8	0.05	0.29	0.03	0.32	NA
33	0.20	0.00	0.00	66	47	12.2	0.12	0.28	0.12	0.31	NA
34	0.21	0.00	0.00	70	42	12.0	0.12	0.29	0.13	0.31	NA
35	0.22	0.00	0.00	41	35	9.8	0.13	0.32	0.13	0.35	NA
36	0.23	0.00	0.00	21	25	8.0	0.17	0.30	0.17	0.32	NA
37	0.26	0.00	0.00	38	21	8.5	0.15	0.38	0.14	0.41	NA
38	0.23	0.00	0.00	96	35	12.2	0.10	0.35	0.10	0.39	NA
39	0.18	0.00	0.00	141	52	15.0	0.08	0.28	0.07	0.31	NA
40	0.15	0.00	0.02	107	53	13.6	0.05	0.25	0.02	0.28	NA
41	0.14	0.00	0.01	74	55	12.3	0.05	0.24	0.04	0.27	NA
42	0.11	0.03	0.05	55	46	10.5	0.01	0.21	0.00	0.24	NA
43	0.08	0.14	0.17	95	53	12.9	-0.02	0.18	-0.04	0.21	NA
44	0.07	0.05	0.09	232	76	20.2	0.00	0.13	-0.01	0.15	NA
45	0.04	0.16	0.30	292	77	22.4	-0.02	0.11	-0.04	0.12	NA
46	0.02	0.63	0.80	156	63	17.0	-0.07	0.11	-0.09	0.12	NA
47	0.02	0.69	0.85	138	60	15.9	-0.07	0.10	-0.10	0.12	NA
48	0.02	0.69	0.95	222	73	22.3	-0.06	0.09	-0.09	0.10	NA
49	0.02	0.63	0.87	204	75	22.9	-0.06	0.10	-0.09	0.10	NA
50	0.03	0.39	0.49	132	65	19.1	-0.04	0.11	-0.06	0.13	NA

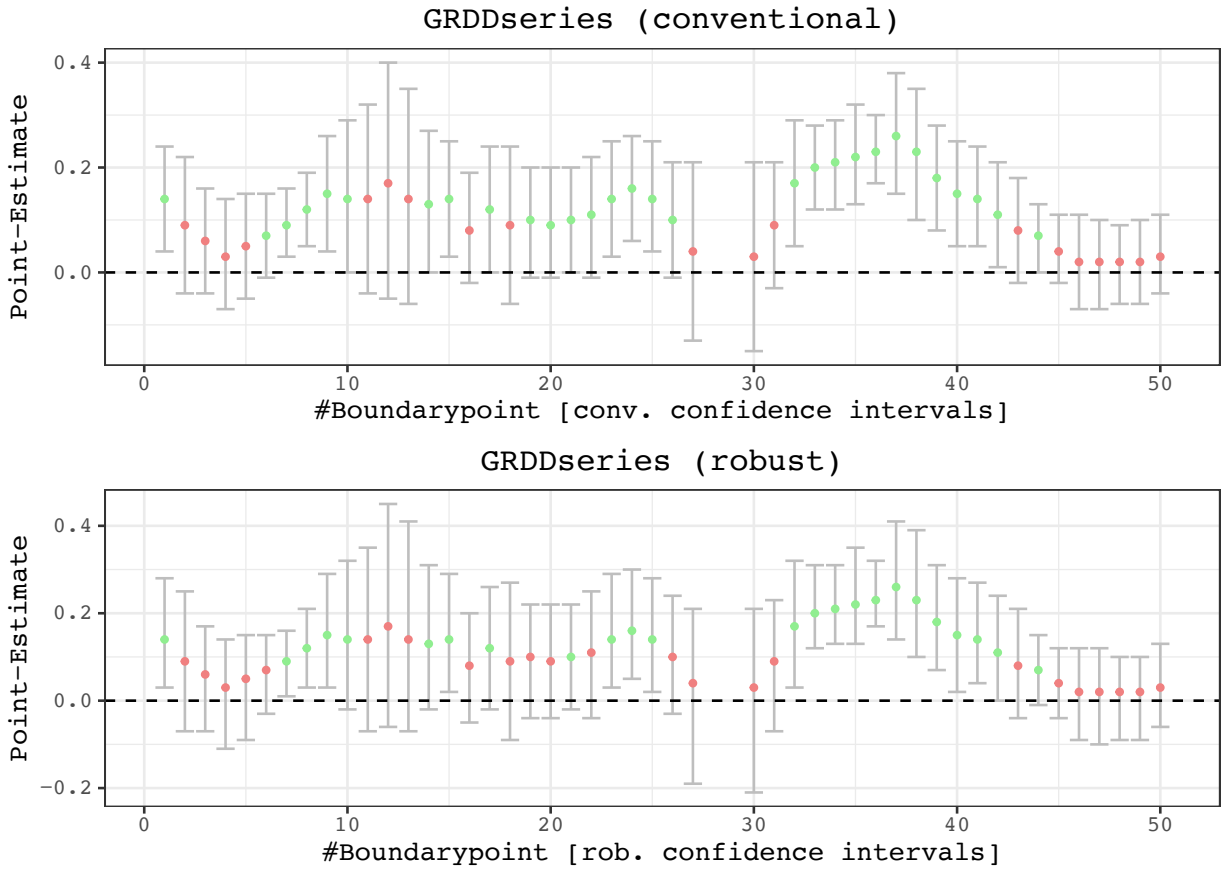
For the plotting of the *GRDDseries* and a visualisation in space of each point estimate we need to have a spatial object. All this is incorporated in the `plotspatialrd()` function.

```
results <- SpatialRD(y = "education", data = points_samp.sf, cutoff.points = borderpoints.sf, treated =
#> We have 1011 observations of which 218 are treated observations.
#> We are iterating over 50 Boundarypoints.
#> The dependent variable is education .
plotspatialrd(results, map = T)
```



just the *GRDDseries*

```
plotspatialrd(results, map = F)
```



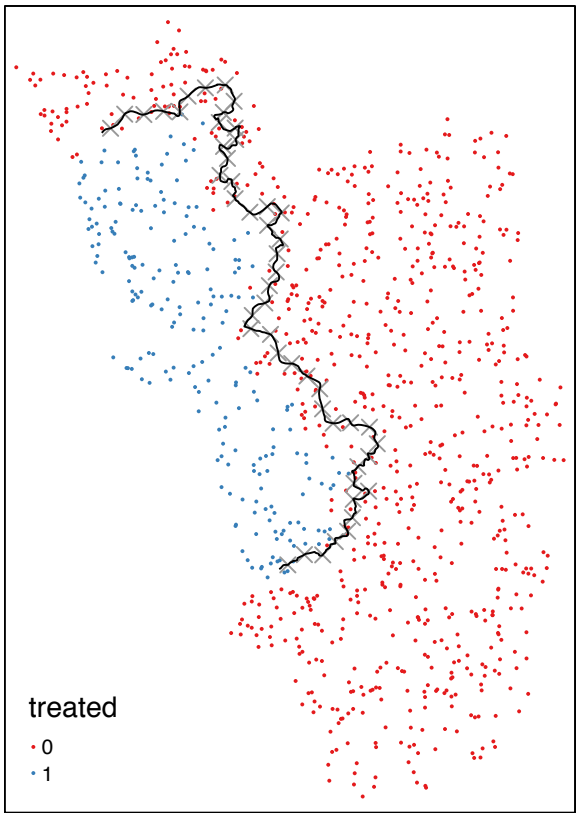
Robustness

In Spatial Regression Discontinuity exercises, the researcher usually also has to show that the results are robust towards different specifications and parameters. Also in this respect the `SpatialRDD` package

Placebo Borders

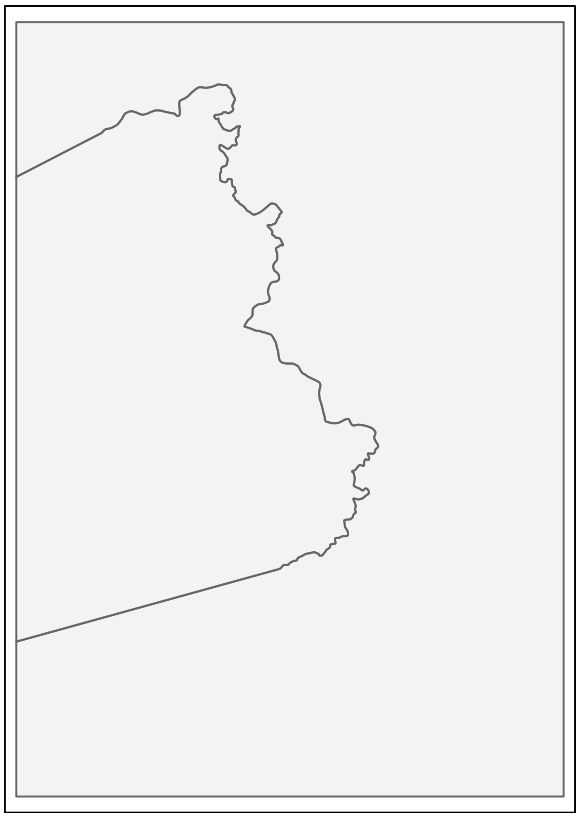
Here we are going to apply a standard tool that we got to know in linear algebra 1 classes: an affine transformation of the type $f(x) = x\mathbf{A} + b$, where the matrix $\mathbf{A}$ is the projection matrix to shift, (re-)scale, or rotate the border. For simplicity we now only apply a shift by 3000 metres in both directions of the border. See the next section for a complete exploration of the `placebo_border()` function and all its capabilities.

```
placebocut_off.1 <- placebo_border(cut_off.sf, operation = "shift", shift = c(3000, 3000))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
placeboborderpoints.1 <- discretise_border(cutoff = placebocut_off.1, n = 50)
#> Starting to create 50 borderpoints. Approximately every 3 kilometres we can run an estimation then.
tm_shape(points_samp.sf) + tm_dots("treated", palette = "Set1")  + tm_shape(placeboborderpoints.1) + tm
```



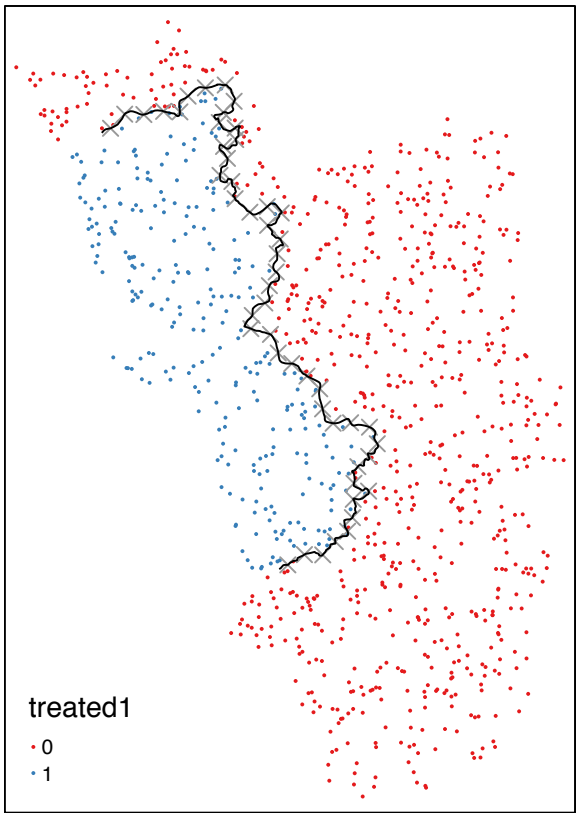
After the border shift we now have to re-assign the new "treated" status in order to carry out regressions. For that matter we create new polygons from scratch with the `cutoff2polygons()` function.

```
polySideUp <- cutoff2polygons(data = points_samp.sf, cutoff = placebocut_off.1, orientation = "West-West
                              crs = 32643, upside = T)
polySideDown <- cutoff2polygons(data = points_samp.sf, cutoff = placebocut_off.1, orientation = "West-W
                                crs = 32643, upside = F)

tm_shape(polySideUp) + tm_polygons(alpha = .3) + tm_shape(polySideDown) + tm_polygons(alpha = .3)
```



Finally we have to use the `assign_treated()` function from the beginning of the vignette again:

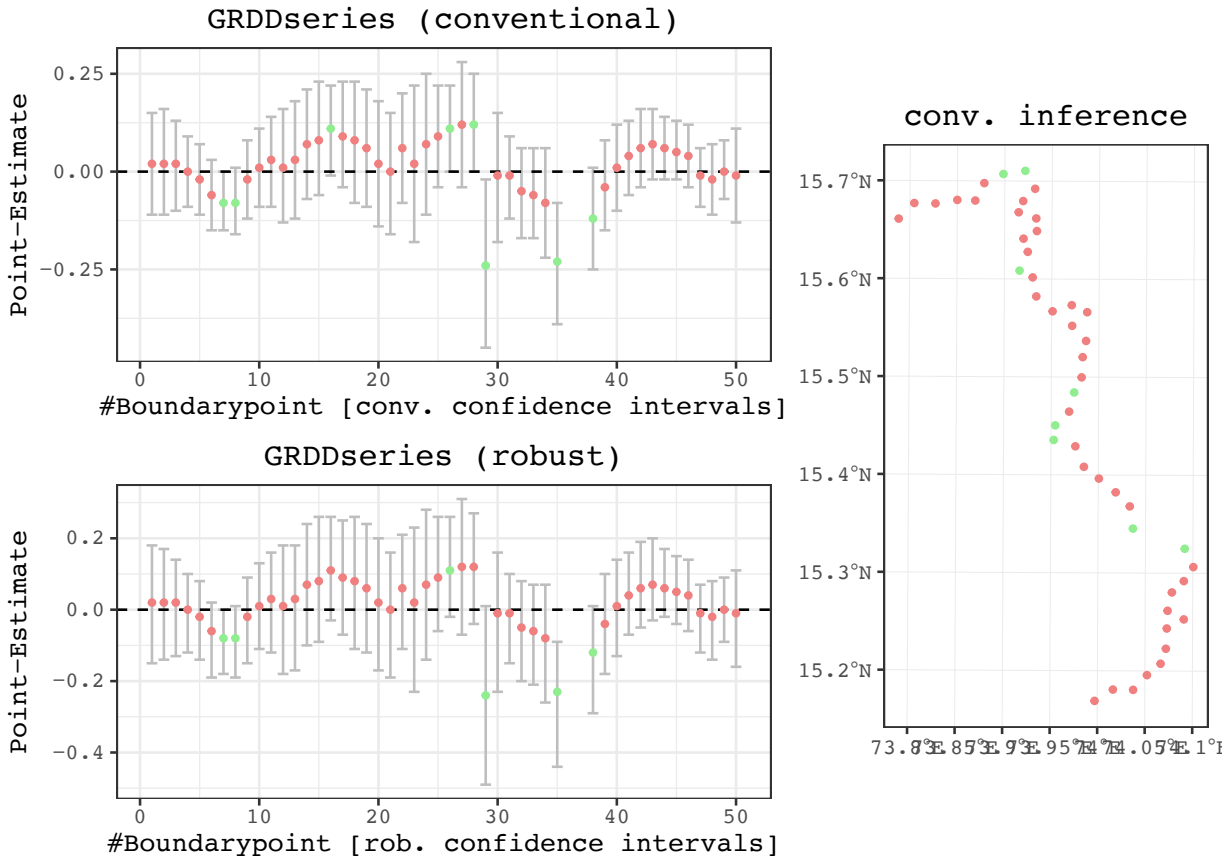
```
points_samp.sf$treated1 <- assign_treated(data.sf = points_samp.sf, polygon.sf = polySideDown, id = "id
#> Warning: attribute variables are assumed to be spatially constant
#> throughout all geometries
sum(points_samp.sf$treated == 0 & points_samp.sf$treated1 == 1) # number of villages that switches
#> [1] 62
tm_shape(points_samp.sf) + tm_dots("treated1", palette = "Set1")  + tm_shape(placeboborderpoints.1) + tm
```



After plotting the points again, we can visually infer that the right villages got assign the "treated" dummy. Further we can compute the number of villages that change their status. This helps us to decide whether the bordershift was big enough (if e.g. only a handful of observations switched, then we would expect this to have little to no impact on our point estimates and thus would dub such a robustness exercise as not very meaningful).
In this case 62 villages changed. Given the initial number of treated observations, this seems a change of a big enough magnitude and thus a meaningful robustness exercise.
Robustness GRD

Running the GRD

```
results1 <- SpatialRD(y = "education", data = points_samp.sf, cutoff.points = placeboborderpoints.1, tr
#> We have 1011 observations of which 276 are treated observations.
#> We are iterating over 50 Boundarypoints.
#> The dependent variable is education .
plotspatialrd(results1, map = T)
```



Robustness Polynomial Specification

Robustness with the polynomial specification. Even this specification is insignificant

```
points_samp.sf$segment.1.5 <- border_segment(points_samp.sf, placebocut_off.1, 5)
#> Starting to create 5 border segments with an approximate length of 26 kilometres each.
points_samp.sf$dist2cutoff1 <- as.numeric(sf::st_distance(points_samp.sf, placebocut_off.1))

summary(lm(education ~ treated1, data = points_samp.sf[points_samp.sf$dist2cutoff1 < 3000, ]))
#>
#> Call:
#> lm(formula = education ~ treated1, data = points_samp.sf[points_samp.sf$dist2cutoff1 <
#>     3000, ])
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.255031 -0.067078 -0.004262  0.065334  0.262374
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  0.60696    0.01035  58.621   <2e-16 ***
#> treated11    0.01416    0.01494   0.948    0.344
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

```
#> Residual standard error: 0.09985 on 177 degrees of freedom
#> Multiple R-squared:  0.005055,   Adjusted R-squared:  -0.0005667
#> F-statistic: 0.8992 on 1 and 177 DF,  p-value: 0.3443
summary(felm(education ~ treated1 | factor(segment.1.5) | 0 | 0, data = points_samp.sf[points_samp.sf$di
#>
#> Call:
#>    felm(formula = education ~ treated1 | factor(segment.1.5) | 0 |      0, data = points_samp.sf[poi
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.226929 -0.071540 -0.004883  0.070360  0.290477
#>
#> Coefficients:
#>           Estimate Robust s.e t value Pr(>|t|)
#> treated11  0.01263    0.01512   0.835    0.405
#>
#> Residual standard error: 0.09752 on 173 degrees of freedom
#> Multiple R-squared(full model): 0.07247   Adjusted R-squared: 0.04566
#> Multiple R-squared(proj model): 0.004188   Adjusted R-squared: -0.02459
#> F-statistic(full model, *iid*):2.703 on 5 and 173 DF, p-value: 0.02222
#> F-statistic(proj model): 0.6976 on 1 and 173 DF, p-value: 0.4047
```

More on placebo bordering

When the border is not approximating a line in space but is curving and bending (i.e. in most cases), "placebo bordering" can be tricky and is not straightforward. The simple substraction/addition of a specified distance from the `distance2cutoff` variable is also not a very accurate description of placebo boundary. On top of that with such a simple transformation of the distance column we can at best do a placebo check on the "pooled" polynomial specification as the border segments change and thus the assignment of categories for fixed effects. A placebo GRD design with an iteration over the boundarypoints is literally impossible in such a case.
For a proper robustness check we thus have to create a new cut-off from which we then can extract the corresponding borderpoints (with `discretise_border()`) and also assign the border segment categories for fixed effects estimations (with `border_segment()`).
The `placebo_border()` function can execute three different (affine) transformations at the same time:

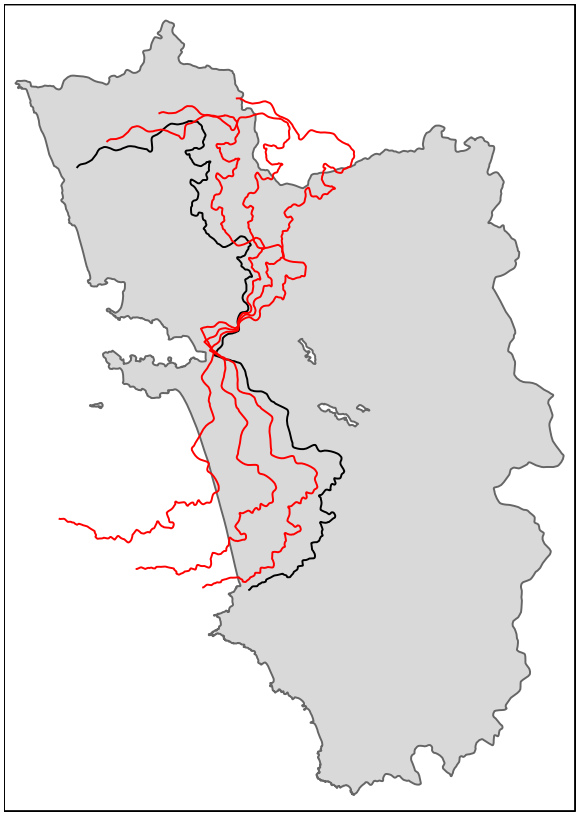
- `"shift"` in units of CRS along the x- & y-axis (provided with the option `shift = c(dist1, dist2)`)
- `"scale"` in percent around the centroid where `0.9` would mean 90%
- `"rotate"` in degrees around the centroid with the standard rotation matrix

$$rotation = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Rotate

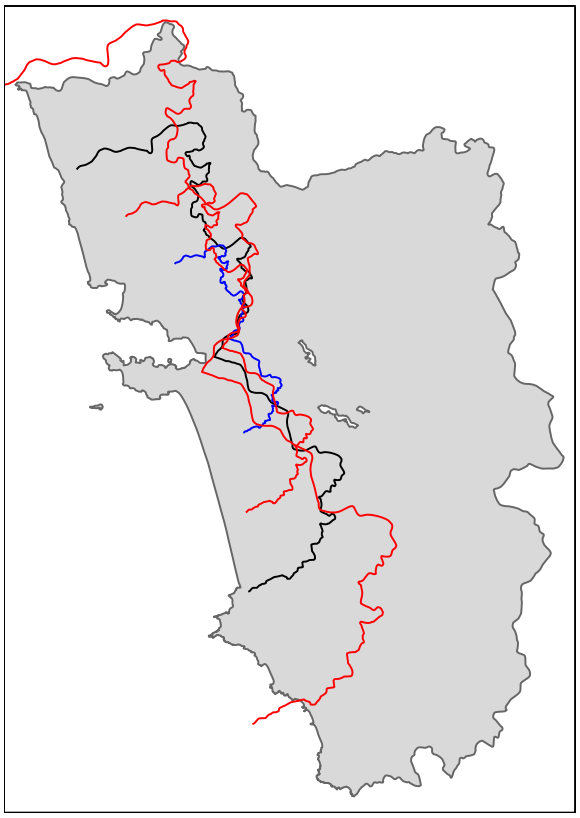
```
tm_rotate.sf10 <- placebo_border(border = cut_off.sf, operation = "rotate", angle = 10)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_rotate.sf25 <- placebo_border(border = cut_off.sf, operation = "rotate", angle = 25)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_rotate.sf45 <- placebo_border(border = cut_off.sf, operation = "rotate", angle = 45)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
```

```
tm_shape(polygon_full.sf) + tm_polygons() + tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(tm_rotate.sf10) + tm_lines(col = "red") +
  tm_shape(tm_rotate.sf25) + tm_lines(col = "red") +
  tm_shape(tm_rotate.sf45) + tm_lines(col = "red")
```



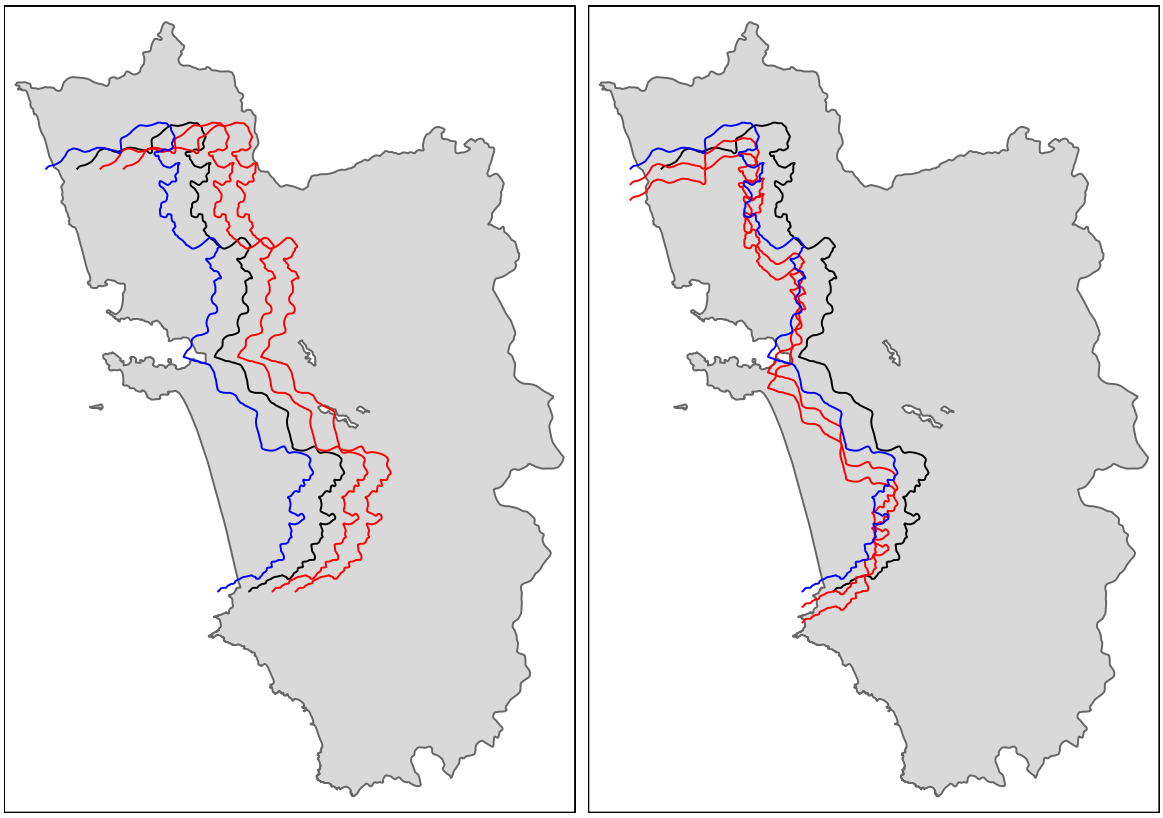
Scale

```
tm_scale.sf.4 <- placebo_border(border = cut_off.sf, operation = "scale", scale = .4)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_scale.sf.7 <- placebo_border(border = cut_off.sf, operation = "scale", scale = .7)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_scale.sf1.5 <- placebo_border(border = cut_off.sf, operation = "scale", scale = 1.5)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l

tm_shape(polygon_full.sf) + tm_polygons() + tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(tm_scale.sf.4) + tm_lines(col = "blue") +
  tm_shape(tm_scale.sf.7) + tm_lines(col = "red") +
  tm_shape(tm_scale.sf1.5) + tm_lines(col = "red")
```



Shift

```
tm_shift.sf3 <- placebo_border(border = cut_off.sf, operation = "shift", shift = c(3000, 0))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_shift.sf6 <- placebo_border(border = cut_off.sf, operation = "shift", shift = c(6000, 0))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_shift.sf_4 <- placebo_border(border = cut_off.sf, operation = "shift", shift = c(-4000, 0))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l

tm_shape(polygon_full.sf) + tm_polygons() + tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(tm_shift.sf3) + tm_lines(col = "red") +
  tm_shape(tm_shift.sf6) + tm_lines(col = "red") +
  tm_shape(tm_shift.sf_4) + tm_lines(col = "blue")

tm_shift.sf_42 <- placebo_border(border = cut_off.sf, operation = "shift", shift = c(-4000, -2000))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_shift.sf_44 <- placebo_border(border = cut_off.sf, operation = "shift", shift = c(-4000, -4000))
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l

tm_shape(polygon_full.sf) + tm_polygons() + tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(tm_shift.sf_42) + tm_lines(col = "red") +
  tm_shape(tm_shift.sf_44) + tm_lines(col = "red") +
  tm_shape(tm_shift.sf_4) + tm_lines(col = "blue")
```

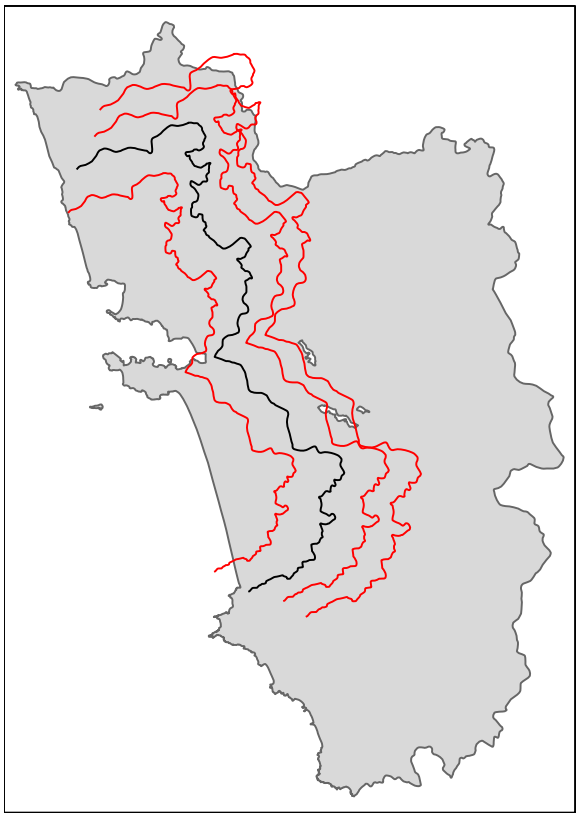


From the last shifted line we can already see that a movement along the x-axis quite often requires also a correction on the y-axis for the cut-off movement to be meaningful. This is going to be explored in detail in the following section together with all the other operations.

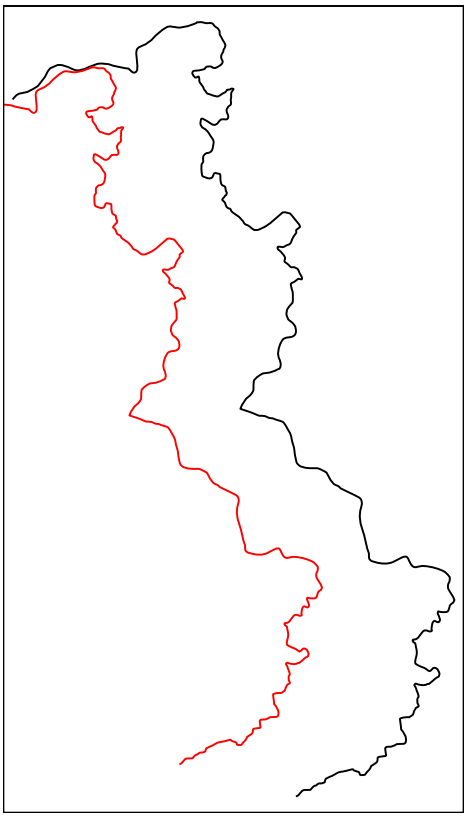
Full fledged "placebo bordering"

A proper placebo border ideally involves both a shift and a re-scaling for it to be meaningful.

```
tm_placebo.sf1 <- placebo_border(border = cut_off.sf, operation = c("shift", "scale"), shift = c(-5000,
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_placebo.sf2 <- placebo_border(border = cut_off.sf, operation = c("shift", "scale"), shift = c(4000, 1
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_placebo.sf3 <- placebo_border(border = cut_off.sf, operation = c("shift", "scale"), shift = c(6000, 3
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_shape(polygon_full.sf) + tm_polygons() + tm_shape(cut_off.sf) + tm_lines() +
  tm_shape(tm_placebo.sf1) + tm_lines(col = "red") +
  tm_shape(tm_placebo.sf2) + tm_lines(col = "red") +
  tm_shape(tm_placebo.sf3) + tm_lines(col = "red")
```

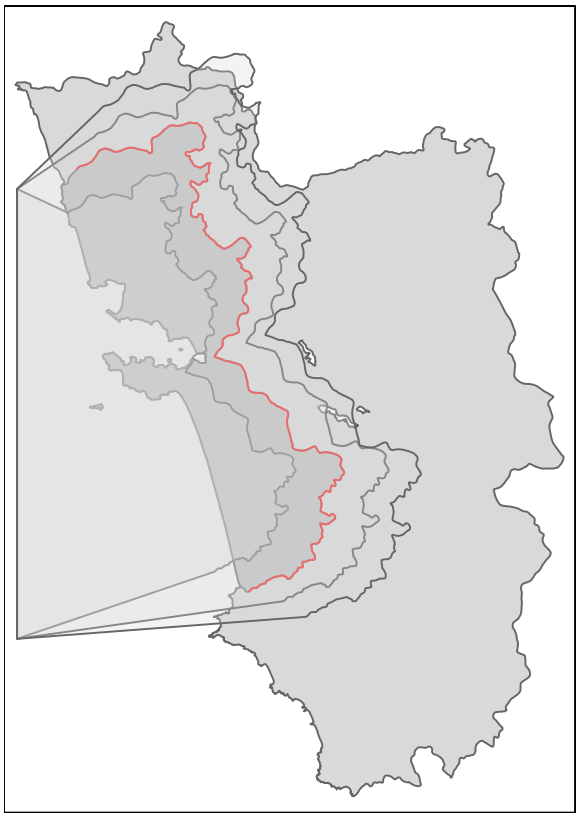


```
tm_shift.sf <- placebo_border(border = cut_off.sf, operation = c("shift", "rotate", "scale"),
                              shift = c(-10000, -1000), angle = 0, scale = .9)
#> Pay attention to CRS! If in 4326 then degrees have to be provided. For precision we would prefer a l
tm_shape(cut_off.sf) + tm_lines() + tm_shape(tm_shift.sf) + tm_lines(col = "red")
```



And the according polygons to assign the treated dummies:

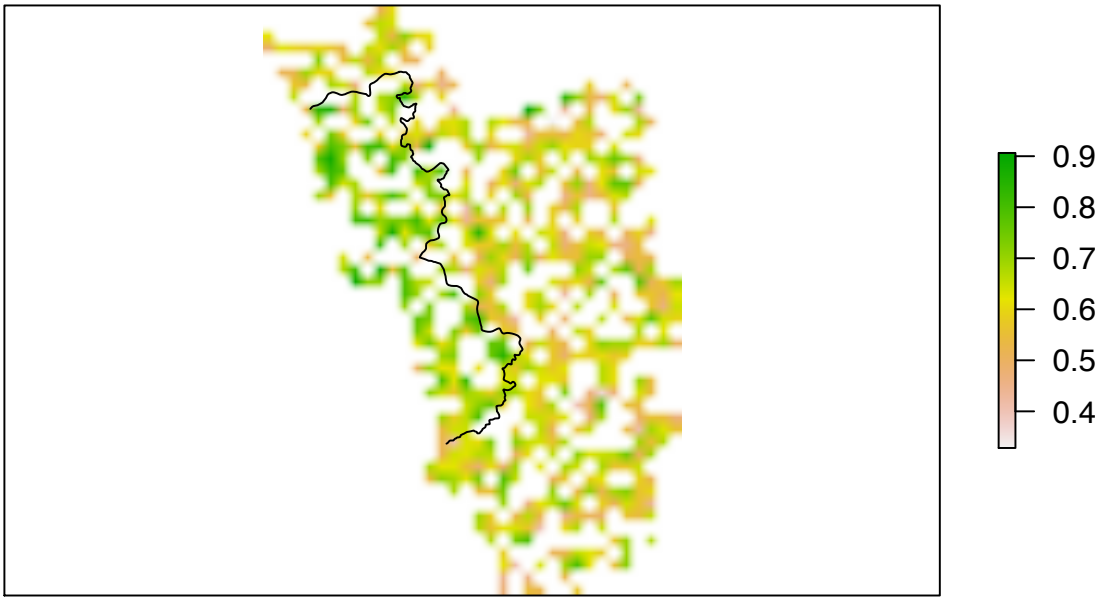
```
polygon1 <- cutoff2polygons(data = points_samp.sf, cutoff = tm_placebo.sf1, orientation = "West-West",
                            crs = 32643, upside = F)
polygon2 <- cutoff2polygons(data = points_samp.sf, cutoff = tm_placebo.sf2, orientation = "West-West",
                            crs = 32643, upside = F)
polygon3 <- cutoff2polygons(data = points_samp.sf, cutoff = tm_placebo.sf3, orientation = "West-West",
                            crs = 32643, upside = F)

tm_shape(polygon_full.sf) + tm_polygons() +
  tm_shape(polygon_treated.sf) + tm_polygons(col = "grey") +
  tm_shape(cut_off.sf) + tm_lines(col = "red") +
  tm_shape(polygon1) + tm_polygons(alpha = .3) +
  tm_shape(polygon2) + tm_polygons(alpha = .3) +
  tm_shape(polygon3) + tm_polygons(alpha = .3)
```



Spatial Predictions

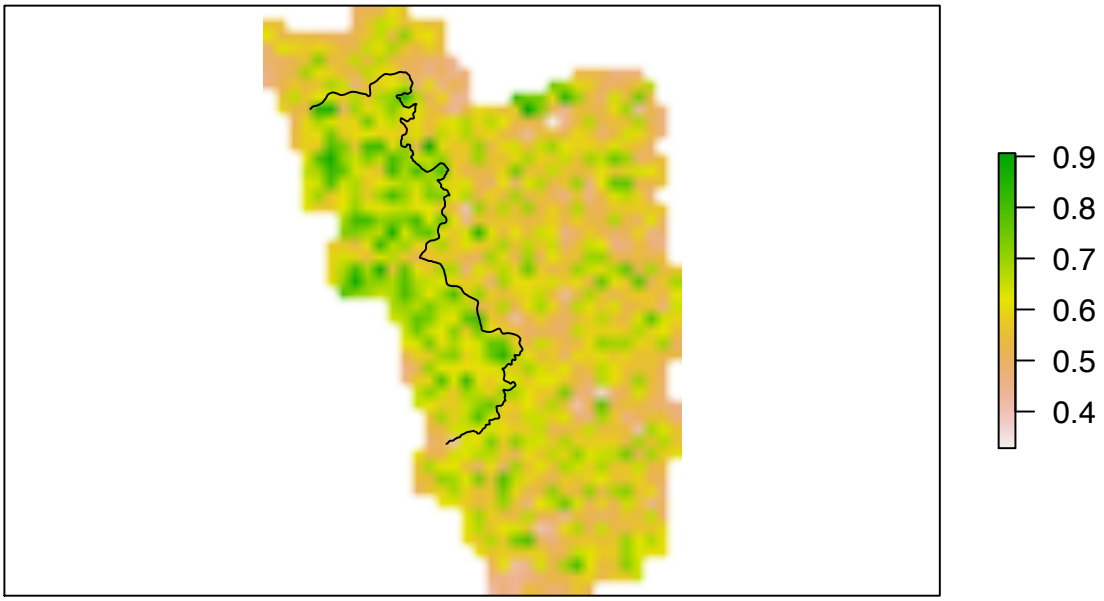
Another, more visual and straightforward, way to convince the audience of a potential discontinuity across space is by interpolating the variables of interest across space. This can be seen as a supplementary way for any Spatial Regression Discontinutiy design that makes the estimation on "placebo boundaries" more or less obsolete, as one can visually infer that the discontinuity is actually happening at the exact cut-off that has been put forward.
After definining the raster template we just compute the mean of the variable of interest across the aribitrary grid-cell size that we chose:

```
library(raster)
raster_template = raster(extent(points_samp.sf), resolution = 2000,
                         crs = st_crs(points_samp.sf)$proj4string)
raster_mean <- rasterize(points_samp.sf, raster_template, field = "education", fun = mean)
plot(rasterize(points_samp.sf, raster_template, field = "education", fun = mean),
     axes = F)
lines(as(cut_off.sf, "Spatial")) # converting the cut-off to sp format for plotting with base-R
```



To also obtain values for neighbouring cells that do not contain any observations, we **manually** choose a simple weighting function, based on 3 cells around and a weight of 0.9:

```
plot(focal(raster_mean, matrix(.9, nc = 3, nr = 3), fun = mean, NAonly = T, na.rm = T, pad = T),
     axes = F)
lines(as(cut_off.sf, "Spatial")) # converting the cut-off to sp format for plotting with base-R
```



Here we just made use of very basic and standard spatial interpolation techniques that can be easily carried out with `raster`, a spatial workhorse package in **R**. One could think of many more sophisticated ways such as kriging.