Blogroll

- Junkcharts
- OECD insights
- R-bloggers

Dostępność lekarzy specjalistów, tutorial

10 paź
2013

Dwa dni temu, w tym wpisie przedstawialiśmy grafiki przygotowane przez Michała Kurtysa w ramach wakacyjnego projektu.

Pan Michał przygotował też krótki tutorial wyjaśniający jak samodzielnie zrobić takie wykresy w R. Poniżej wklejamy ten tutorial. Jest on moim zdaniem bardzo ciekawy i porusza wiele ciekawych technicznych problemów z analizą danych przestrzennych.

Dane
Informacje o kolejkach zapisane są w plikach Excelowych o rozszerzeniu xls.
Na każde województwo przypada takich plików kilka. Nas teraz będzie interesował tylko plik dotyczący świadczeń specjalistycznych, zawierający w nazwie skrót AOS.
W sumie jest więc 16 plików (po jednym na województwo), które wyglądają mniej-więcej w ten sposób.

```
01_AOS_31072013.xls
02_AOS_31072013.xls
...
16_AOS_31072013.xls
```

Województwa są ułożone alfabetycznie – najmniejszy numer odpowiada województwu dolnośląskiemu, a największy zachodniopomorskiemu. Aby powiązać numer z nazwą województwa przygotowałem plik **województwa.csv**, w którym są one wypisane w należytej kolejności.

```
województwa.csv
WOJ. DOLNOŚLĄSKIE
WOJ. KUJAWSKO-POMORSKIE
WOJ. LUBELSKIE
WOJ. LUBUSKIE
WOJ. ŁÓDZKIE
WOJ. MAŁOPOLSKIE
WOJ. MAZOWIECKIE
WOJ. OPOLSKIE
WOJ. PODKARPACKIE
WOJ. PODLASKIE
WOJ. POMORSKIE
WOJ. ŚLĄSKIE
```

```
WOJ. ŚWIĘTOKRZYSKIE
WOJ. WARMIŃSKO-MAZURSKIE
WOJ. WIELKOPOLSKIE
WOJ. ZACHODNIOPOMORSKIE
```

Dane o granicach administracyjnych Polski pobrałem z Geoportalu. Ze względów licencyjnych początkowo chciałem wykorzystać OpenStreetMap. Gotowe pliki shapefile oferuje m.in. geofabrik/cloudmade. Niestety, jeżeli chodzi o Polskę nie miałem tam czego szukać – wszystkie mapy były zdeformowane.
Mapy z Geoportalu nie są dostępne w formacie shapefile. Na szczęście dzięki instrukcjom we wpisie Pawła Wiechuckiego przedstawianego na tym blogu ich własnoręczne stworzenie nie było trudne.

Biblioteki
Potrzebujemy następujących bibliotek:
** maptools – funkcja readShapePoly pozwala na wczytanie pliku ShapeFile.
** sp – do manipulacji i wyświetlania danych kartograficznych.
** rgeos, rgdal – funkcja spTransform, pozwala zmienić układ współrzędnych
** FNN – funkcja get.knnx – do regresji
** SmarterPoland – funkcję getGoogleMapsAddress ułatwi pobranie współrzędnych punktu o określonym adresie
** Cairo – do produkcji wykresów
** XLConnect – obsługa plików excela

Kod R

Zacznijmy od wczytania bibliotek.

```
1 library(maptools)
2 library(sp)
3 library(rgeos)
4 library(SmarterPoland)
5 library(FNN)
6 library(rgdal)
7 library(Cairo)
8 library(XLConnect)
```

Następnie zmiana katalogu roboczego. Wczytanie listy województw.

Z każdego pliku "aos" odczytujemy dane, które zaczynają się w 3 wierszu(razem z nagłówkiem) i mają 9 kolumn.
Postawnowiłem zmienić nazwy kolumn, gdyż oryginalne były niesłychanie długie.
Dodatkowo tworzymy dodatkowe kolumny – ID i nazwę województwa.

```
 9 setwd("C:\\Users\\mic\\Documents\\eurostat\\nfzqueue\\new")
10 wojewodztwa = read.csv("wojewodztwa.csv", header=FALSE)
11 wojewodztwa$V1 = as.character(wojewodztwa$V1)
12
13 poradnie = data.frame()
14 for( i in 1:16) {
15         filename = sprintf("%02d_AOS_31072013.xls", i)
16         print(filename)
17         wb = loadWorkbook( filename, create = FALSE)
18         dane = readWorksheet(wb,sheet="Zestawienie", startRow = 3, startCol=0, endCol=8)
19         dane$IDWojewodztwa = i
20         dane$Wojewodztwo = wojewodztwa$V1[i]
21         poradnie = rbind(poradnie, dane)
22 }
23
24 #zmiana nazwy kolumn
```

```
names(poradnie) = c(
"Nazwa.komorki.realizujacej",
"Kategoria",
"Nazwa",
"Nazwa.komorki",
"Adres",
"Liczba.oczekujacych",
"Liczba.skreslonych",
"Sredni.czas",
"ID.wojewodztwa",
"Wojewodztwo")
```

Pacjenci są podzieleni przez NFZ na dwie grupy – “przypadek stabilny” i “przypadek ostry”.
Graficznie przedstawiać będziemy wyłącznie dane o przypadkach stabilnych.

Dalej dzielimy adres na części składowe – nazwa miejscowości i ulica razem z numerem numer lokalu.

```
poradnie = poradnie[which(poradnie$Kategoria == "przypadek stabilny"), ]
poradnie$Miejscowosc = sapply(strsplit(poradnie$Adres, "\n"), function(x) x[1] )
poradnie$Ulica = sapply(strsplit(poradnie$Adres, "\n"), function(x) x[2] )
poradnie$Sredni.czas = as.numeric(poradnie$Sredni.czas)
```

Pobieramy współrzędne wyłącznie miejscowości w której znajduje się poradnia.
Jest to znacznie szybsze, a z pewnością nie potrzebujemy większej dokładności.
Po za tym Google ogranicza darmowy dostęp do usługi do 2500 zapytań dziennie.

Konstruujemy więc tabelę zawierającą nazwę miejscowości w jednej kolumnie, a w drugiej województwo w którym się ona znajduje.
Dodanie województwa pomaga uściślić zapytanie – istnieją przecież miejscowości, które noszą tę samą nazwę.

Funkcja unique eliminuje zduplikowane wiersze.
W pętli tworzymy nową zmienną region, w której zapisujemy jedynie nazwę województwa bez skrótu “Woj.” na początku.
Google Geocoding raczej nie trawi tego przedrostka.

Województwo łączymy z nazwą miejscowości i przekazujemy do argumentu city funkcji getGoogleMapAddress.
Może to wyglądać to dziwnie, ale zapytanie i tak zostanie skonstruowane poprawnie.

```
places_unique = data.frame( poradnie$Miejscowosc, poradnie$Wojewodztwo, check.names=FALSE)
places_unique = unique( places_unique )
names(places_unique) = c("Miejscowosc", "Wojewodztwo")
places_unique$Wojewodztwo = as.character(places_unique$Wojewodztwo)
places_unique$Miejscowosc = as.character(places_unique$Miejscowosc)

for( i in 1:nrow(places_unique) ) {
        region = strsplit( places_unique$Wojewodztwo[i], " " )[[1]][2]
        temp_coords = getGoogleMapsAddress(street="",city=paste(places_unique$Miejscowosc[i],",",regio

        places_unique$lat_wgs84[i] = temp_coords[1]
        places_unique$lng_wgs84[i] = temp_coords[2]
}
```

Zawczasu warto przygotować sobie współrzędne w układzie, który jest bardziej przyjazny do zadania:
Tworzymy zmienną typu SpatialPoints i określamy ich układ współrzędnych.

Współrzędne, które pobraliśmy są w układzie odniesienia WGS 84, znanym także jako EPSG:4326

Zmieniamy układ współrzędnych przy pomocy funkcji spTransform.

Ostatecznie łączymy dwie tabele przy pomocy funkcji merge.
Wreszcie mamy wszystkie interesujące nas wartości.

```
53 places_sp = SpatialPoints( cbind(places_unique$lng_wgs84, places_unique$lat_wgs84) )
54 proj4string(places_sp) = CRS("+init=epsg:4326")
55 places_sp = spTransform( places_sp, CRS("+init=epsg:2180"))
56
57 places_unique$lng = coordinates(places_sp)[,1]
58 places_unique$lat = coordinates(places_sp)[,2]
59
60 poradnie = merge(poradnie, places_unique)
```

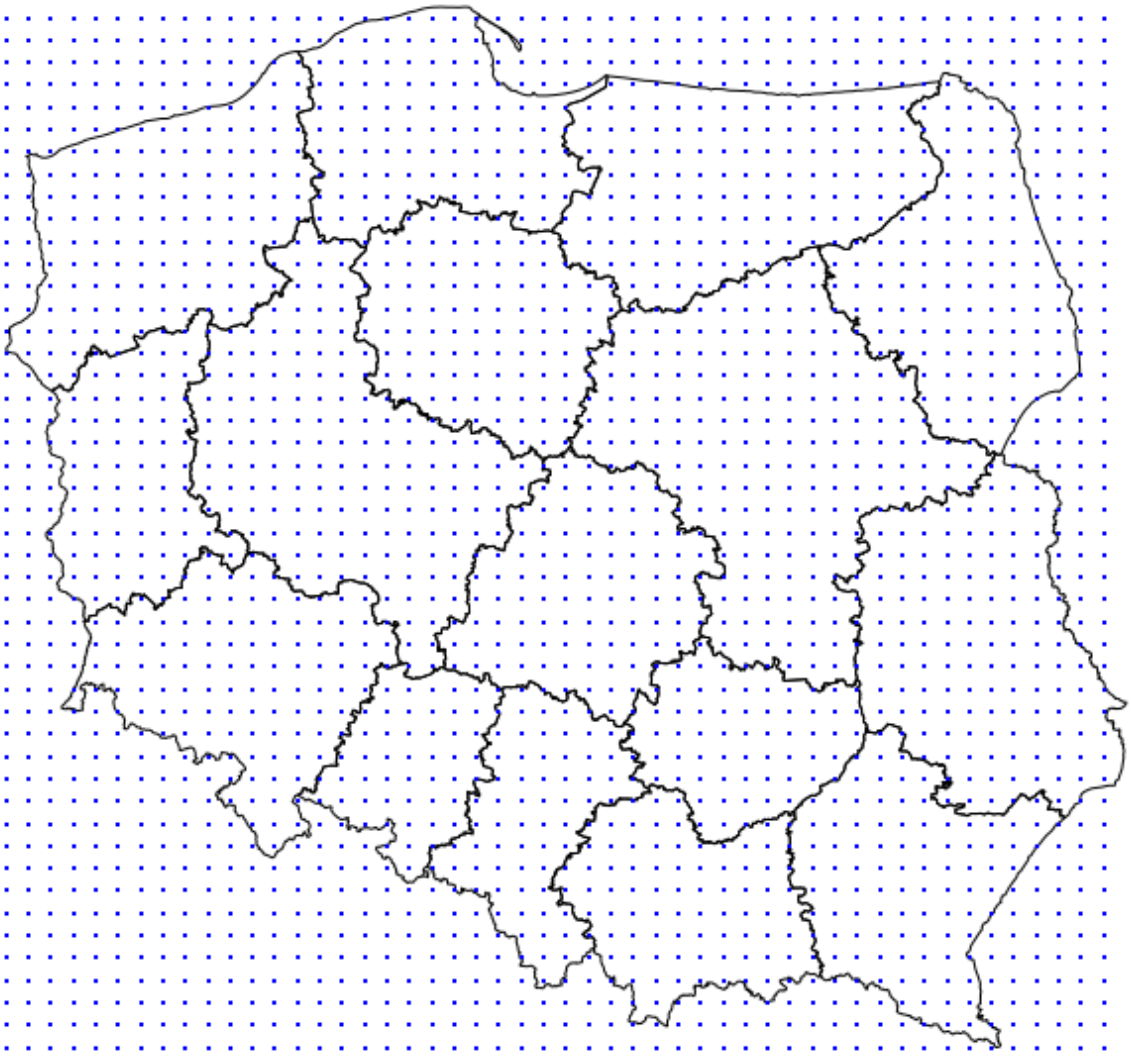
Wczytanie pliku shapefile, który zawiera granice administracyjne województw.
Jest już w prawidłowym układzie współrzędnych – wystarczy tylko go określić.

```
61 wojewodztwa.shp = readShapePoly("geoportal\\woj.shp")
62 proj4string(wojewodztwa.shp)=CRS("+init=epsg:2180")
```

Funkcja generate_grid posłuży do wygenerowania kraty, która będzie pokrywać powierzchnię określonego pola.
Na obrazku widać punkty kraty. W istocie są to punkty, które będą służyły jako punkty startowe linii.

Pierwszy argument funkcji to ilość komórek kraty w osi X.
Komórka kraty ma być kwadratem, więc ich ilość w osi Y nie będzie przekazywana do funkcji.

Następnie wyliczamy topologię kraty. Przekazujemy:
** współrzędne dolnego, lewego rogu
** wymiary komórek
** ilość komórek
I na koniec zwracamy obiekt klasy SpatialGrid.



```
63 generate_grid = function( cell_n_x=1000, sp_polygons) {
64         bbox_min = bbox(sp_polygons)[,1]
65         bbox_max = bbox(sp_polygons)[,2]
66         cell_width=((bbox_max-bbox_min)/cell_n_x)[1]
67         cell_height=cell_width
68         cell_n_y = ceiling(((bbox_max-bbox_min)/cell_height)[2])
69 
70         gt = GridTopology( bbox_min, c(cell_width,cell_height), c(cell_n_x,cell_n_y)  )
71         sg = SpatialGrid(gt, CRS("+init=epsg:2180"))
72         return(sg)
73 }
```

Jak widać na obrazku, część punktów kraty znajduje się poza granicami kraju.
Funkcja which_cells_inside zwraca indeksy komórek kraty, które znajdują się wewnątrz obiektu SpatialPolygons.

Wykorzystamy do tego funkcję over. Dla argumentów SpatialPoints i SpatialPolygons funkcja zwraca wektor o długości równej ilości punktów.
Wartości wektora mówią o tym, wewnątrz którego wielokąta znajduje się punkt.
Tych mamy 16 – odpowiadają województwom.

Jeżeli jest poza jakimkolwiek wielokątem składowym to danemu punktowi odpowiadać będzie wartość NA.

Szczegółowo wielokąt składowy to obiekt klasy Polygons:

```
> class(wojewodztwa.shp)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
> class(wojewodztwa.shp@polygons)
[1] "list"
> class(wojewodztwa.shp@polygons[[1]])
[1] "Polygons"
attr(,"package")
[1] "sp"
> length(wojewodztwa.shp@polygons)
[1] 16
```

```
which_cells_inside = function( sp_grid, sp_polygons) {
        grid_coords = coordinates(sp_grid)
        grid_points = SpatialPoints(grid_coords, CRS( proj4string(sp_grid)) )
        inside = over( grid_points, sp_polygons)
        proj4string(grid_points) = proj4string(wojewodztwa.shp)
        inside = which( inside$ID1 > 0 )
        return(inside)
}
```

Generowanie linii. W argumentach znajduje się zmienna cell_distance, która mówi co ile komórek kraty będzie znajdować się początek linii.
Gdy pisałem kod uznałem, że być może takie rozwiązanie się przyda. Teraz jestem pewien, że bardziej elegancko byłoby generować mniejszą kratę.
Co z resztą każdy zauważy.

Zwracamy obiekt typu SpatialLines. Strukura obiektów (Spatial)Line(s) jest podobna do Polygons.

Przypominam, że linie rysujemy od punktu kraty do najbliższej przychodni.
Musimy więc wiedzieć, która jest najbliższa.
Przekazujemy więc macierz knn_indices w której będzie to zapisane.

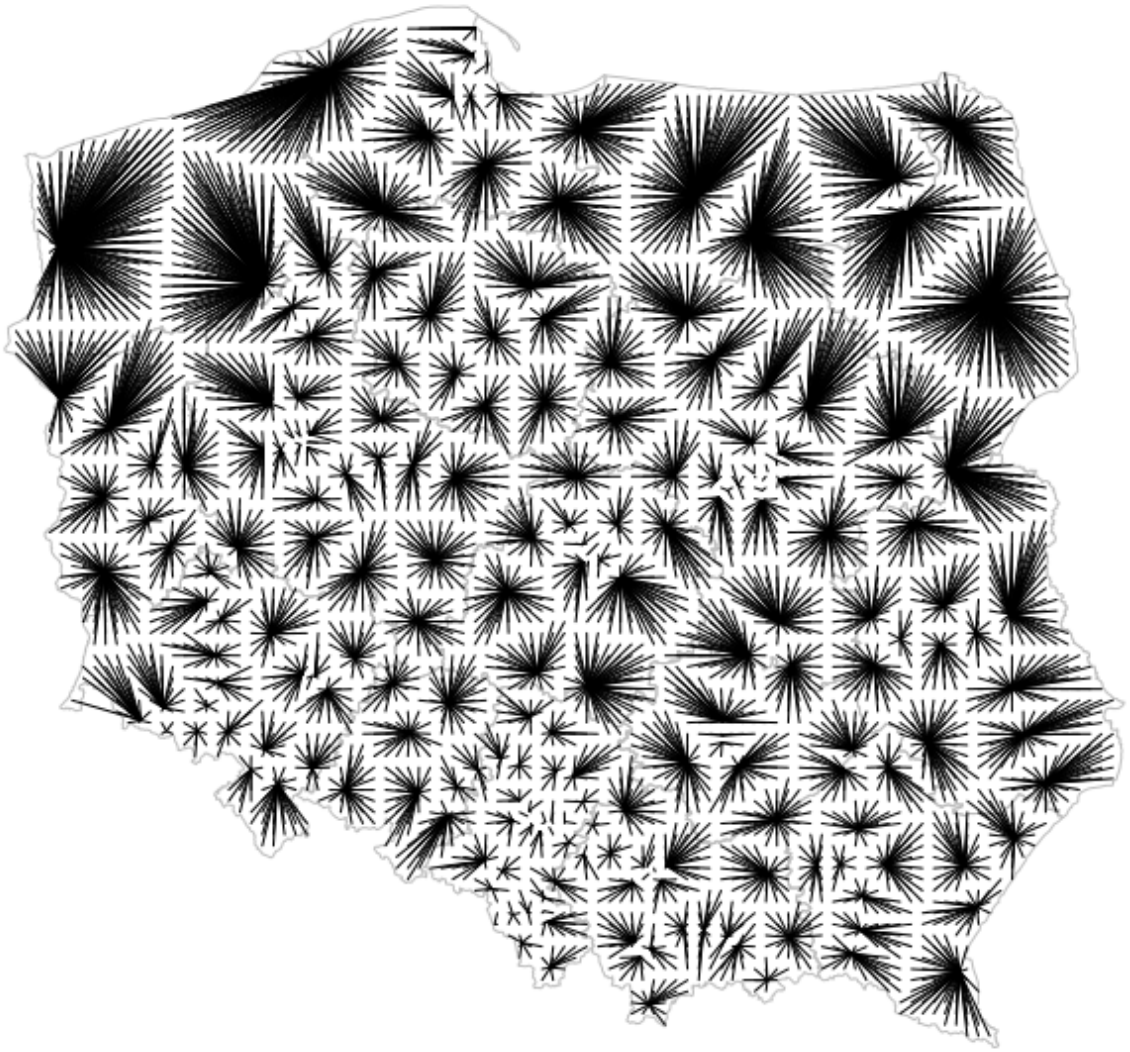
```
# sp_points to obiekt typu SpatialPoints, zawierający listę przychodni.
# cell_subset ogranicza kratę, tak by linie były prowadzone tylko z wewnątrz kraju.

generate_lines = function( knn_indices, sp_grid, sp_points, cell_distance=1, cell_subset=NULL ) {
        cell_n_x = sg@grid@cells.dim[1]
        cell_n_y = sg@grid@cells.dim[2]
        #macierz ktora zawiera indeksy komorek w ktorych zaczynaja sie linie
        line_anchors = matrix(nrow=floor(cell_n_y/cell_distance), ncol=floor(cell_n_x/cell_distance))
        for( i in 1:nrow(line_anchors)) {
                line_anchors[i,] = seq(cell_distance, cell_n_x, by= cell_distance) +
                                                rep(cell_n_x*cell_distance*i,floor(cell_n_x/cell_di
        }
        #tylko te linie ktorych poczatek jest w wybranych komorkach
        #na przyklad wewnatrz kraju
        line_anchors = intersect(line_anchors, cell_subset)

        #konstrukcja obiektow biblioteki sp
        line_starts = coordinates(sp_grid)[line_anchors,]
        closest_medic_idx = knn_indices[line_anchors,1]

        line_stops = coordinates(sp_points)[closest_medic_idx,]
        line_list = sapply( 1:nrow(line_starts), function(x) Line( rbind( line_starts[x,], line_stops
        line_obj = Lines(line_list, ID="a")
        sp_lines_obj = SpatialLines( list(line_obj) )
```

```
118
119         return(sp_lines_obj)
120 }
```

Odpalamy funkcje.
W pętli przejdziemy po kolei po ka¿dym typie poradni.
Wewnątrz niej wybierzemy interesujące nas wiersze w tabeli poradnie.
Dzięki bibliotece Cairo łatwo zapiszemy wykres w wysokiej rozdzielczości.
get.knnx jest funkcją obsługującą regresję k-sąsiadów.
W pierwszym argumencie jest zbiór danych, w drugim zbiór zapytań.
W tej sytuacji interesuje nas wyłącznie najbliższy sąsiad, więc k wynosi 1.
Zwraca dwie macierze – w pierwszej zapisuje indeksy najbliższych sąsiadów, a w drugiej odległości.

```
123 #wczytanie danych, etc..
124 #kod
125 sg = generate_grid( cell_n_x=100, wojewodztwa.shp)
126 inside_poland = which_cells_inside( sg, wojewodztwa.shp)
127
128 for( specialist_type in unique( (poradnie$Nazwa.komorki.realizujacej) ) ) {
129         print(specialist_type)
130         flush.console()
131         specialist = poradnie[which(poradnie$Nazwa.komorki.realizujacej == specialist_type),]
132         specialist_sp = SpatialPoints( cbind( specialist$lng, specialist$lat) ,CRS("+init=epsg:2180")
133
134         knn_res = get.knnx( specialist_sp@coords, coordinates(sg), k = 1 )
135
136         sp_lines_obj = generate_lines( knn_res[[1]], sg, specialist_sp, cell_distance=1, cell_subset=
137
138         Cairo(900, 700, file=paste(specialist_type,".png",sep=""), type="png", bg="white")
139         plot(wojewodztwa.shp)
140         plot(sp_lines_obj,add=T)
141         #plot(specialist_sp,col="blue",lwd=2, add=T)
142         title(main=specialist_type)
143         dev.off()
144 }
```

PORADNIA PULMONOLOGICZNA

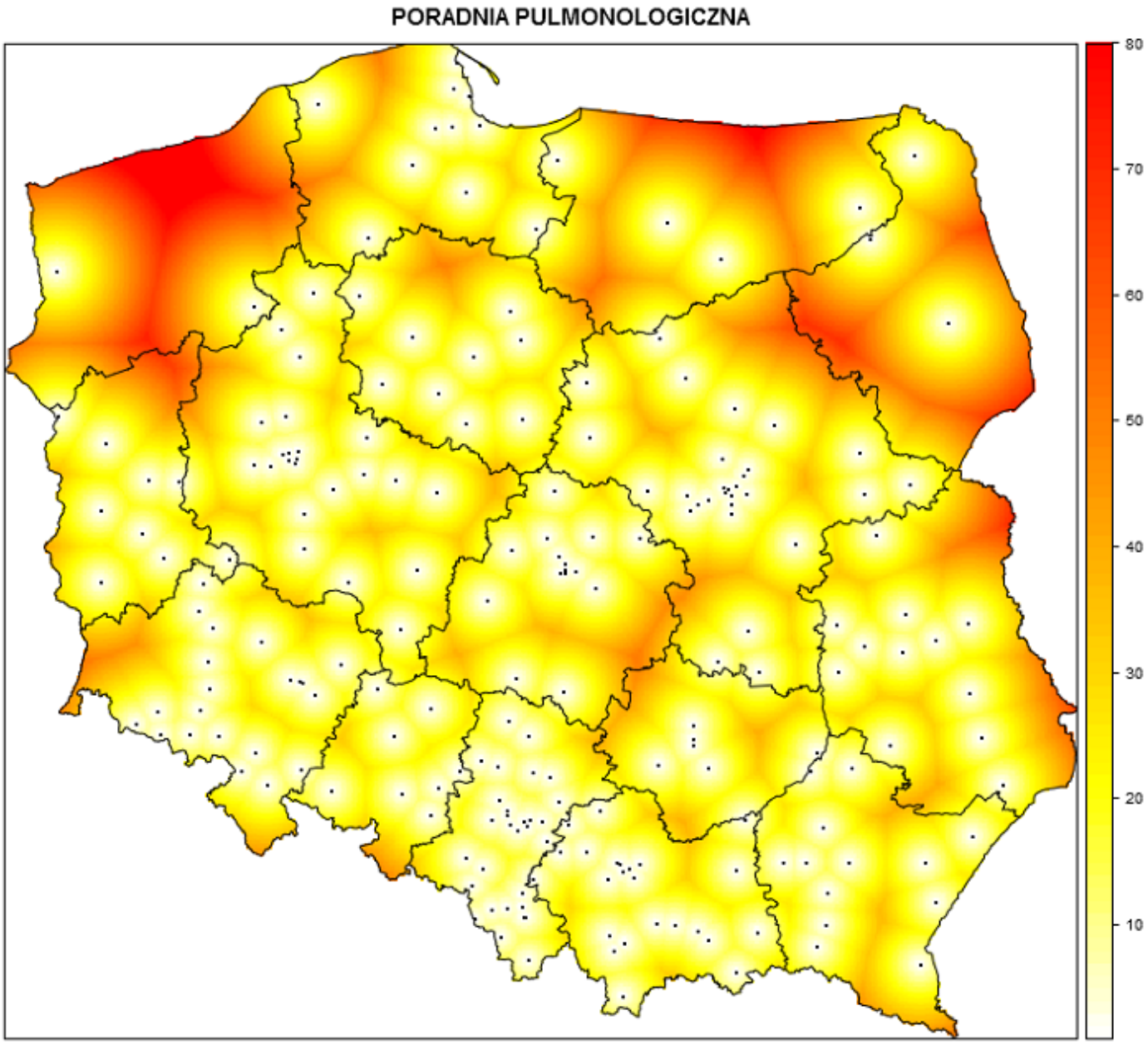


Hidden track. Tworzenie map “kolorowych”. Tworzymy gęstszą kratę.
Następnie uzyskujemy punkty kraty i zmieniamy ich układ współrzędnych do WGS 84.

Jest to niezbędne – dla układu współrzędnych epsg:2180 funkcja spDistsN1 nie zwracała odległości w kilometrach.

```
145 sg = generate_grid( cell_n_x=500, wojewodztwa.shp)
146 inside_poland = which_cells_inside( sg, wojewodztwa.shp)
147
148 grid_points_wgs84 = SpatialPoints( coordinates(sg), CRS(proj4string(sg)))
149 grid_points_wgs84 = spTransform(grid_points_wgs84, CRS("+init=epsg:4326"))
150 gp_coords = coordinates(grid_points_wgs84)
151
152 for( specialist_type in unique( (poradnie$Nazwa.komorki.realizujacej) ) ) {
153         print(specialist_type)
154         flush.console()
155         specialist = poradnie[which(poradnie$Nazwa.komorki.realizujacej == specialist_type),]
156         specialist_sp = SpatialPoints( cbind( specialist$lng, specialist$lat) ,CRS("+init=epsg:2180")
157         specialist_coords_wgs84 = cbind( specialist$lng_wgs84, specialist$lat_wgs84)
158
159         #knn_res = get.knnx( specialist_sp@coords, coordinates(sg), k = ifelse(nrow(specialist)>10, 1
160         knn_res = get.knnx( specialist_sp@coords, coordinates(sg), k = 1)
```

```
161
162         knn_indices = knn_res[[1]]
163         km_distances =  matrix( nrow=nrow( knn_indices ), ncol=ncol(knn_indices))
164         closest_coords = apply( knn_indices, c(1,2), function(x) specialist_coords_wgs84[x,])
165         closest_coords = matrix( closest_coords, ncol = 2, byrow=TRUE ) #2*ncol(knn_indices)
166
167         for (rowID in 1:nrow( knn_indices ) ) {
168                 pts = matrix(closest_coords[rowID,],ncol=2)
169
170                 p = gp_coords[rowID,]
171                 km_distances[rowID,] = spDistsN1(pts, p, TRUE)
172         }
173         closest_distance = km_distances[,1]
174         sg_df = SpatialGridDataFrame( sg, data.frame(closest_distance))
175
176         #obszar położony poza granicami kraju ma kolor biały
177         sg_df@data$closest_distance[-inside_poland]=0
178         #80 jest górną granicą wartości na wykresie
179         sg_df@data$closest_distance[ sg_df@data$closest_distance>80] = 80
180
181         Cairo(900, 700, file=paste(specialist_type,"_c",".png",sep=""), type="png", bg="white")
182         print(spplot( sg_df, panel = function(x,y,...){
183         panel.gridplot(x,y,...,  )
184         sp.polygons( wojewodztwa.shp )
185         sp.points( specialist_sp, pch=".", col="black" )
186         }, at=c(1:80), col.regions = rev(heat.colors(100)), main=specialist_type))
187         dev.off()
188
189 }
```



Było to mój pierwszy „większy” projekt w R. Z pewnością wiele rozwiązań nie jest najlepszych, ale mimo wszystko mam nadzieję, że ten wpis będzie pomocny.

Edit this entry.

- In: R|Społeczeństwo|Służba Zdrowia|Wizualizacja
- Tags: kod R, kolejki do specjalistów, mapa, NFZ

Comment Form

Logged in as smarterpoland