Ćwiczenia 10

Temat: Zanieczyszczenie powietrza w Krakowie cz.4.

Co będziemy robić: sporządzenie mapy stężenia pyłów PM2.5.

Dane do ćwiczenia.

1. Dane z serwisu airly (https://developer.airly.org/pl), po zalogowaniu dostajecie klucz, który służy do pobrania danych.
2. Paczka z konturami dzielnic Krakowa znajduje się w na kanale głównym w zakładce pliki/Materiały z zajęć

Najpierw utworzymy sobie strukturę danych, do której ostatecznie dołączymy dane dotyczące zanieczyszczenia z czujników (musimy znowu powtórzyć część kroków, które już znamy).

```
> install.packages("httr") # do obsługi połączeń internetowych
> library(httr)
> install.packages("jsonlite")
> library(jsonlite)
```

Pobieramy dane o czujnikach w odległości 15km od ratusza:

```
> r <-
GET("https://airapi.airly.eu/v2/installations/nearest?lat=50.0617022&lng=19
.9373569&maxDistanceKM=15&maxResults=-1",
        add_headers(apikey = "xxxxx", Accept = "application/json")
)
#xxxxx – tu wpisujemy klucz dostępu do danych
```

Przejście do listy:

```
> jsonRespText<-content(r,as="text")
> test15<-fromJSON(jsonRespText)
```

Tworzymy ramkę z danymi o lokalizacji czujników (długość i szerokość geograficzna, wysokości n.p.m.), dodamy też id czujników:

```
> longitude<-test15$location$longitude
> latitude<-test15$location$latitude
> data15<-data.frame(longitude,latitude)
> data15$elevation<-test15$elev #wysokość co prawda nie będzie potrzebna, ale niech będzie, jak poprzednio
> data15$id<-test15$id
> head(data15)
```

Tworzymy obiekt przestrzenny (ppp).

```
> install.packages("sp")
> library(sp)
> install.packages("spatstat")
> library(spatstat)
> data_spat<-
data.frame(lon=data15$longitude,lat=data15$latitude,elev=data15$elev,id=dat
a15$id)
> coordinates(data_spat) <- ~lon+lat #określamy, które elementy to koordynaty
(potrzebne do ppp)
> proj4string(data_spat) <- CRS("+proj=longlat +datum=WGS84") #określamy,
jaki mamy układ
> data_spat # teraz mamy już obiekt w układzie sferycznym, który można automatycznie konwertować
```

Konwertujemy do UTM (bo tworzymy ppp, a to jego układ).

```
> data_UTM <- spTransform(data_spat, CRS("+proj=utm +zone=34
+datum=WGS84"))
```

Aby utworzyć obiekt ppp, ale tylko z czujnikami w Krakowie, potrzebny będzie obiekt krakowUTM, którym przytniemy data_UTM :

```
> install.packages("sf")
> library(sf)
> dzielnice<-st_read("dzielnice_Krakowa.shp")
> dzielniceWGS84<-st_transform(dzielnice,crs = 4326)
> krakowWGS84<-st_union(dzielniceWGS84) #zostawiamy tylko kontur miasta
> krakowUTM<-st_transform(krakowWGS84,CRS("+proj=utm +zone=34
+datum=WGS84"))
```

Zaraz utworzymy obiekt ppp z danymi ("marks") w punktach (uwaga: dane to id), przyciętymi do granic Krakowa oknem krakowUTM. Jeszcze tylko utworzymy wcześniej zmienną pomocniczą z koordynatami (bo spTransform wprowadziło swoje nazwy kolumn z koordynatami):

```
> XY<-coordinates(data_UTM)
```

I obiekt ppp 2D:

```
> data15_ppp_id<-
ppp(x=data_UTM$lon,y=data_UTM$lat,marks=data.frame(elev=data_UTM$elev,id=da
ta_UTM$id),window=as.owin(krakowUTM))
```

```
> data15_ppp_id$marks$id #dzięki przycięciu konturem mamy od razu tylko id tych czujników, które są w Krakowie
```

########pobieranie danych z Airly dla wybranych czujników (czyli tych z data15_ppp_id)########

Za chwilę zaciągniemy dane z wybranych właśnie czujników.
Wcześniej jednak musimy przygotować dwa obiekty, tj. liczbę czujników oraz ich id:

```
> n_id<-length(data15_ppp_id$marks$id)
> id<-data15_ppp_id$marks$id
```

Tworzymy też pustą listę o długości n_id, w której później znajdą się dane z czujników:

```
> list_inst2 <- vector(mode = "list", length = n_id)
```

Teraz, korzystając z pętli pobierzemy dane z wybranych czujników Airly (czujnik to ”installation”).

```
> for (i in seq(1,n_id)) {

print(i) #to dla nas tylko informacja, w którym obrocie pętli jesteśmy

str<-
paste("https://airapi.airly.eu/v2/measurements/installation?installationId=
",id[i],sep="") # określamy adres, pod którym są pomiary z konkretnego czujnika
r <- GET(url=str, add_headers(apikey = "xxxxx", Accept =
"application/json"))
jsonRespText<-content(r,as="text")
inst<-fromJSON(jsonRespText) #przejście do tekstowej formy

list_inst2[[i]]<-inst # tutaj będą zapisywane wszystkie odczyty

}
```

Funkcja paste służy do tworzenia ciągu znaków (string) z wyszczególnionych obiektów według schematu.
xxxxx – tu wpisać trzeba, jak poprzednio, swój klucz

Uwaga: Na wszelki wypadek warto zapisać pełną listę do pliku (bo dziennie możemy wykonać tylko 100 zapytań do Airly, jak coś pójdzie źle, to trzeba będzie czekać 24h na kolejną możliwość ściągnięcia danych):

```
> save(list_inst2,file="list_inst2.Rdata") # wczytanie danych poleceniem load()
```

Wypiszmy informacje o wybranym czujniku (tutaj drugi ze stworzonej listy, pierwszy akurat nie działał kiedy tworzyłam materiały):

```
> list_inst2[[2]]
```

Jak widać, struktura danych jest wysoce nieprzyjemna :(, ale nas na razie interesuje tylko pole current dla PM 2.5 (w zapisie widoczne jako PM25), czyli dane najświeższe. Oprócz tego mamy

jeszcze inne dane z ostatnich 24 godzin (history) oraz prognozę na następne 24 godziny (forecast).

Problem z danymi jest też taki, że nie każdy czujnik rejestruje taki sam zestaw parametrów (np. któryś może rejestrować stężenia CO_2, NO_2, PM10 oraz PM2.5, tmp., a inny tylko PM 2.5 i tmp.). Przykładowo, zobaczmy różnice dla 1 i 6 czujnika:

```
> list_inst2[[2]]$current$values
     name   value
1         PM1    8.00
2        PM25   12.38
3        PM10   13.81
4    PRESSURE 1006.06
5    HUMIDITY   91.33
6 TEMPERATURE    3.94
```

```
> list_inst2[[6]]$current$values
        name   value
1         PM1    6.31
2        PM25   11.15
3        PM10   12.39
4    PRESSURE 1005.19
5    HUMIDITY   91.90
6 TEMPERATURE    3.73
7         NO2   39.16
8          O3   32.64
```

Wybierzemy potrzebne dane – to znaczy odczyty aktualnego (”current”) stężenia PM2.5 zarejestrowanego dla wybranych czujników.
1. Najpierw utworzymy pusty wektor (funkcją rep – do replikacji wartości), o długości n_id, w którym później znajdą się dane PM 2.5 z czujników:

```
> current<-rep(NA,n_id)
```

2. Następnie wyciągamy istniejące wartości current dla PM2.5 (uwaga: PM2.5 oznaczone jako PM25) za pomocą pętli:

```
> for (i in seq(1,n_id)) {

print(i)

logic<-list_inst2[[i]]$current$values$name=="PM25"

if (sum(logic)==1)
current[i]<-list_inst2[[i]]$current$values[logic,2]
}
```

logic # zmienna logiczna służąca do wyszukania pól, o odpowiedniej nazwie, tutaj szukamy PM25 (inne pola to np. preassure, humidity, NO2)
if # testujemy, czy dla danego czujnika istnieje jedno i tylko jedno pole o zadanej nazwie (tutaj PM25). Zdarza się, że pojawiają się błędne odczyty, w których jest wiele pól o takich samych nazwach, nie wiadomo które z nich jest dobre, dlatego takich danych nie zapisujemy do wektora current (pomijamy je). Zobaczmy to - przeanalizujemy pętlę krok po kroku:

```
> i<-1
> logic<-list_inst2[[i]]$current$values$name=="PM25"
> logic
[1] FALSE  TRUE FALSE FALSE FALSE FALSE # jak widać dla czujnika pierwszego jest tylko jedno pole PM25

> list_inst2[[i]]$current$values
         name   value
1         PM1   25.09
2        PM25   40.69
3        PM10   77.36
4    PRESSURE 1024.80
5    HUMIDITY   87.22
6 TEMPERATURE    2.97

> list_inst2[[i]]$current$values[logic,]
  name value
2 PM25 40.69
> list_inst2[[i]]$current$values[logic,2] # wyciągamy tylko wartość
[1] 40.69
```

Zobaczmy teraz utworzony wektor z wartościami PM2.5 czujników zlokalizowanych w Krakowie:

```
> current
 [1]  39.60  36.72     NA  37.09     NA  45.47  32.44  35.88     NA     NA  35.25  55.54  38.54  40.29  43.04
[16]     NA  50.41  38.21     NA     NA  58.59     NA  45.11  49.85  49.76  60.51  45.39  68.19     NA     NA
[31]  45.55  57.85  56.77  56.69  59.55     NA     NA  51.39  51.07  45.17  62.87  68.08  69.59  67.72     NA
[46]  56.53     NA  50.48  48.73  52.92  40.82  45.21  81.99  89.81  66.23  46.27  36.51  45.25  58.89  54.32
[61]  44.78  36.50     NA     NA   8.33     NA  80.49  42.54  57.63     NA  36.54 101.11  59.71     NA  93.68
[76]  83.32     NA  99.82  86.98 109.75  39.84  58.22     NA  34.32 106.29  38.14  55.68  47.00  62.55  75.09
[91] 132.45  60.44 123.84     NA     NA
```

#jak widać nie dla wszystkich czujników mamy wartości PM2.5.

#####Wykonanie mapy#####

Przekształcamy obiekt `data15_ppp_id` do obiektu `spdf` (jak pamiętacie ten format obsługuje funkcja `autoKrige()`, która posłuży do wykonania mapy).

```
> library(automap)

> data15_spdf <- as.data.frame(data15_ppp_id)
```

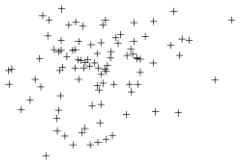
Określamy koordynaty i układ odniesienia:

```
> coordinates(data15_spdf) <- ~x+y

> proj4string(data15_spdf) <- CRS("+proj=utm +zone=34 +datum=WGS84")
```

Dodajmy kolumnę current:

```
> data15_spdf$current<-current
> dev.off()
> plot(data15_spdf)
```

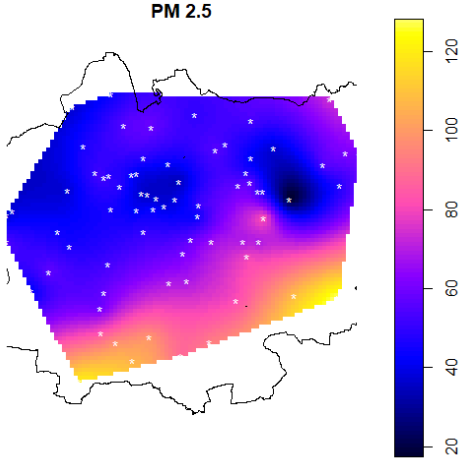


W przeciwieństwie do wysokości, mamy braki w danych dotyczących PM2.5 na czujnikach (current). Funkcja autoKrige nie poradzi sobie sama z wartościami NA, musimy je więc odpowiednio oznaczyć (za pomocą utworzonego wektora logicznego miss):

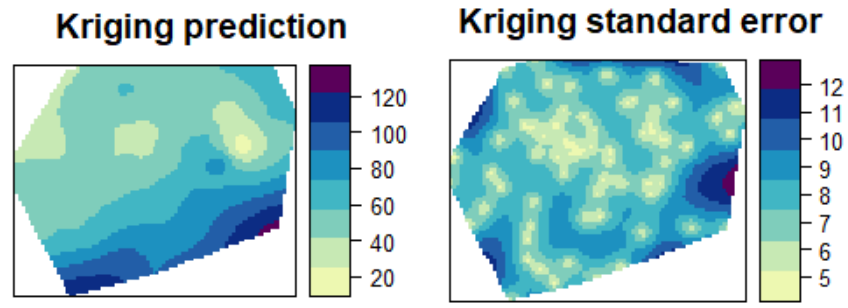
```
> miss <- is.na(data15_spdf$current)
```

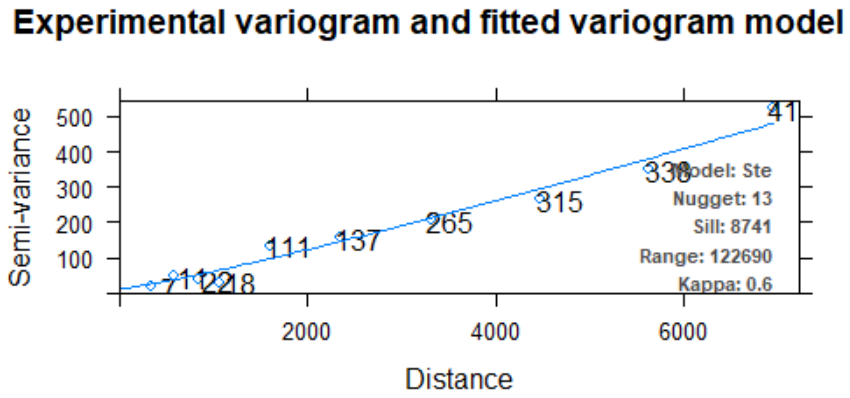
Teraz możemy już wykonać mapę:

```
> pm25_auto <- autoKrige(current ~ 1, input_data = data15_spdf[!miss,])
> plot(pm25_auto$krige_output[1],main="PM 2.5")
> points(data15_ppp_id[!miss,],pch="*",col="White")
> plot(Window(data15_ppp_id),add=TRUE)
```



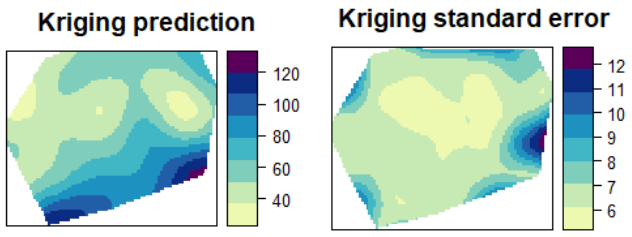
plot(pm25_auto)

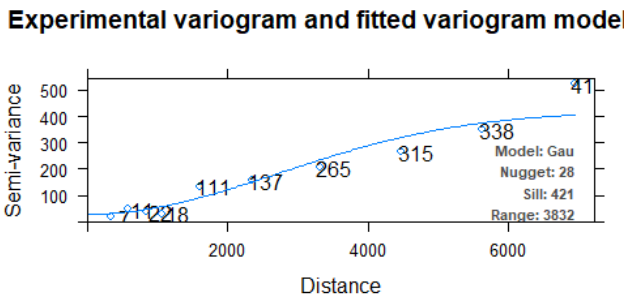




zwróćmy uwagę, że został automatycznie wybrany model `Ste` i wynik wydaje się do przyjęcia. Czasami, kiedy procedura automatyczna zawiedzie (np. wygeneruje ujemne wartości), trzeba model wybrać manualnie. Robimy to przy pomocy parametru `model` w funkcji `autoKrige`, np.:

```
> pm25_auto <- autoKrige(current ~ 1, input_data = data15_spdf[!miss,],
model="Gau")
```





Należy dodać, że nie każdy wybrany przez nas „na sztywno” model (przez zadanie parametru `model`) będzie oczywiście dobry/prawidłowy. Warto jednak zarówno poeksperymentować, jak też

dowiedzieć się więcej na temat możliwych do wyboru modeli (patrz poniżej). Oczywiście, nie każdy wybrany przez nas „na sztywno" model, będzie możliwy do zastosowania na naszych danych - wtedy, po zastosowaniu `autoKrige()` pojawi się komunikat o błędzie.
Oto polecane modele:

- *Matern, M. Stein's parameterization* Ste
- Nuggetowy (ang. *Nugget effect model*) Nug
- Sferyczny (ang. *Spherical model*) Sph
- Gaussowski (ang. *Gaussian model*) Gau
- Potęgowy (ang. *Power model*) Pow
- Wykładniczy (ang. *Exponential model*) Exp
- *Spline* Spl

Aby dowiedzieć się więcej o tych modelach (i nie tylko) warto zaglądnąć tutaj:
http://www.wbc.poznan.pl/Content/382515/Nowosad_Jakub_Geostatystyka_w_R.pdf
(strona 83).
Polecam też krótki filmik, dla przypomnienia koncepcji krigingu:
"The Kriging Model : Data Science Concepts"
https://www.youtube.com/watch?v=J-IB4_QL7Oc
a także filmik tłumaczący co to jest semi-variogram
https://www.youtube.com/watch?v=SJLDlasDLEU&t=315s

#Dla dociekliwych polecam:
https://www.youtube.com/watch?v=jVRLGOsnYuw #wprowadzenie do wariogramu – zobaczyć warto też kolejne części tej serii

oraz
https://www.youtube.com/watch?v=0dAVkBo_hKY&list=PL5B4v-RS9DkWv_5lDo6IaxnZN3p4UrbT8&index=1
#wraz z dalszymi częściami tej serii

Przydatny też jest kurs "Spatial Statistics in R" na na DataCamp.

##
Na koniec zrobimy bardzo ładną mapę zanieczyszczenia Krakowa pyłami PM2.5 ☺ . Sposób jej tworzenia pokazany był już na poprzednich zajęciach, dlatego poniżej znajdują się tylko podstawowe komentarze.
Musimy mieć kontur Krakowa w odpowiednim formacie:

```
> bound<-st_as_sf(krakowUTM)
> plot(bound)
```

Pobieramy współrzędne punktów konturu w formie macierzy:

```
> coord<-as.data.frame(st_coordinates(krakowUTM))
```

Najpierw utworzymy siatkę - prostokąt okalający kontur Krakowa:
1. Określamy współrzędne naroży

```
> left_down<-c( min(coord$X), min(coord$Y))
> right_up<-c( max(coord$X), max(coord$Y))
```

2. Ustalamy rozmiar oczka siatki (100x100 metrów)

```
> size<-c(100,100)
```

3. Obliczamy liczbę oczek siatki przypadających na długość i szerokość prostokąta:

```
> points<- (right_up-left_down)/size
> num_points<-ceiling(points) #zaokrąglenie w górę
```

4. Wreszcie tworzymy siatkę…

```
> grid <- GridTopology(left_down, size,num_points)
```

5. …i konwertujemy ją do odpowiedniego formatu, w odpowiednim układzie (tu: WGS84)

```
> gridpoints <- SpatialPoints(grid, proj4string = CRS("+proj=utm +zone=34
+datum=WGS84"))
> plot(gridpoints) #czekamy cierpliwie
```

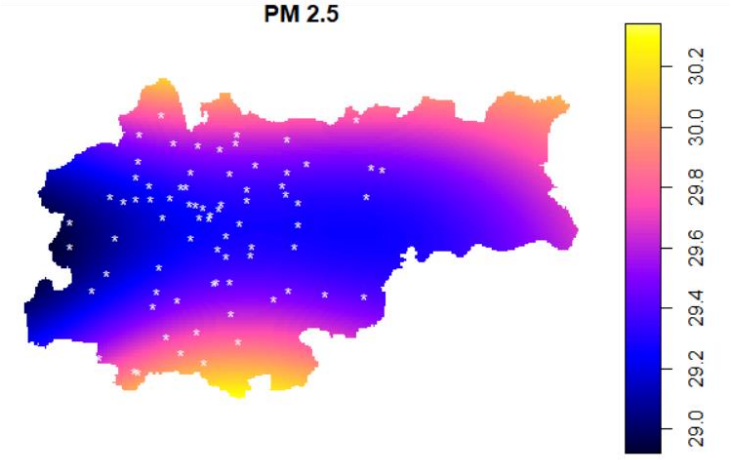
Teraz przycinamy utworzoną siatkę konturem Krakowa funkcją crop_shape z pakietu tmaptools

```
> install.packages("tmaptools")
> library(tmaptools)
> g<-st_as_sf(gridpoints) #konwersja do formatu na którym działa crop_shape
> cg<-crop_shape(g,bound,polygon = TRUE)
> spgrid <- SpatialPixels(as_Spatial(cg)) #konwersja z powrotem do st, a
następnie do SpatialPixels
> plot(spgrid)
```

Rysujemy mapę z wykorzystaniem krigingu:

##uwaga: ”current” zamiast ”marks”!

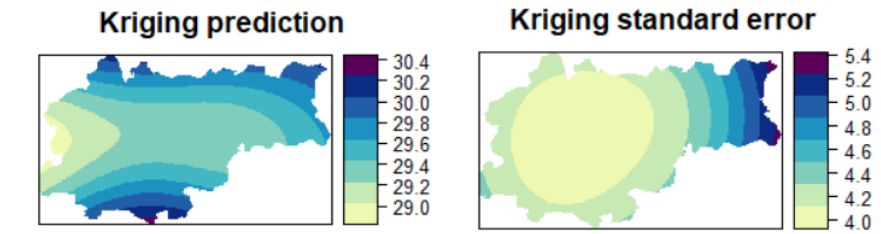
```
> pm25_auto <- autoKrige(current ~ 1, input_data =
data15_spdf[!miss,],new_data=spgrid)
> plot(pm25_auto$krige_output[1],main="PM 2.5")
> points(data15_ppp_id[!miss,],pch="*",col="White")
```

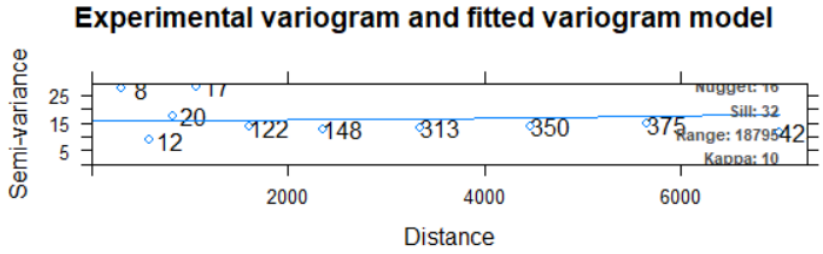


*wykonana mapa bazuje na innych danych PM2.5 niż poprzednia w tych materiałach, prosta mapa (dane pobrane w innym terminie).

Zobaczmy też błędy i semiwariogram:

```
> plot(PM25_auto)
```


Kriging prediction
30.4
30.2
30.0
29.8
29.6
29.4
29.2
29.0
Kriging standard error
5.4
5.2
5.0
4.8
4.6
4.4
4.2
4.0


Experimental variogram and fitted variogram model
Semi-variance
25
15
5
8
17
20
12
122
148
313
350
375
42
Nugget: 16
Sill: 32
Range: 18795
Kappa: 10
2000
4000
6000
Distance