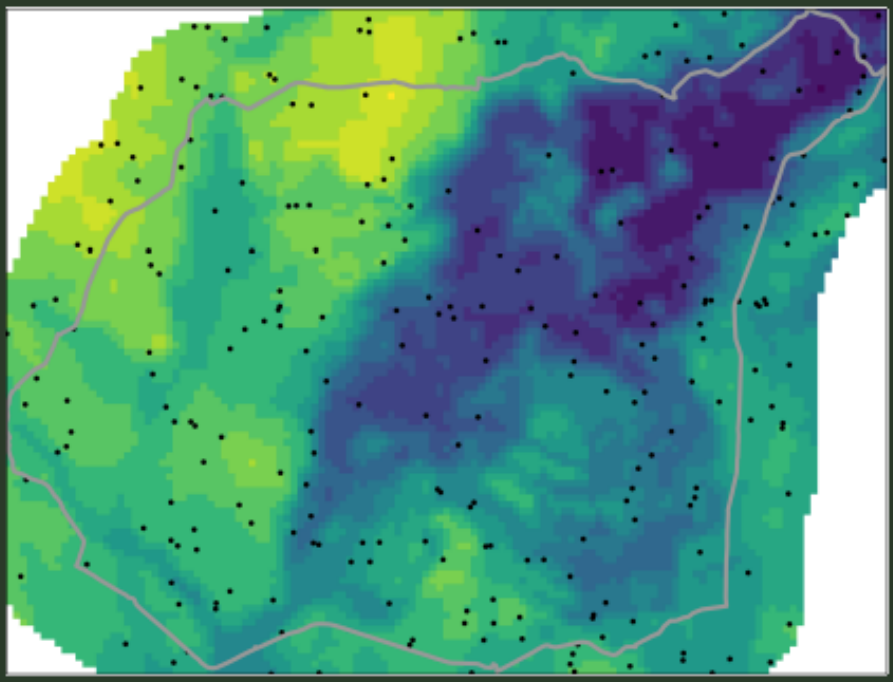
# Geostatystyka w R



Jakub Nowosad

# Geostatystyka w R

Jakub Nowosad

Wydanie drugie, Poznań 2019
Space A

Więcej informacji https://bookdown.org/nowosad/Geostatystyka/

ISBN: 978-83-953296-0-9

# Spis treści

# O skrypcie

Masz przed sobą skrypt zawierający materiały do ćwiczeń z geostatystyki. Składa się ona z kilkunastu rozdziałów pokazujących jak: wygląda geostatystyczna analiza danych (rozdział 1), dodawać i wizualizować dane przestrzenne w R (rozdział 2), wykonywać wstępną eksplorację danych nieprzestrzennych (rozdział 3), wstępnie analizować dane przestrzenne (rozdział 4), wykorzystywać deterministyczne metody interpolacji (rozdział 5), rozumieć i wykorzystywać przestrzenne miary podobieństwa i niepodobieństwa (rozdział 6), modelować semiwariogramy bezkierunkowe i kierunkowe (rozdział 7), tworzyć estymacje jednozmienne (rozdział 8), estymacje wykorzystujące dane uzupełniające (rozdział 9), estymacje wielozmienne (rozdział 10), estymacje danych kodowanych (rozdział 11), oceniać jakość wykonanych estymacji (rozdział 12) oraz budować symulacje przestrzenne (rozdział 13). Począwszy od drugiego, każdy rozdział kończy się również szeregiem zadań, które pozwalają na sprawdzenie umiejętności i ich utrwalenie.

Dodatkowo załączone są cztery appendiksy. W appendiksie A można znaleźć odnośniki do innych materiałów związanych z geostatystyką i R, appendiks B opisuje dane użyte w tym skrypcie, appendiks C wprowadza pojęcie powtarzalnego przykładu i tłumaczy jak taki przykład stworzyć, a appendiks D pokazuje uproszczony przykład analizy geostatystycznej.

Wszystkie zaprezentowane metody i analizy zawierają również kod w języku R. Skrypt został stworzony w R (R Core Team, 2019) z wykorzystaniem pakietów **bookdown** (Xie, 2020a), **rmarkdown** (Allaire et al., 2019), **knitr** (Xie, 2020b) oraz programu Pandoc[1]. Aktualna wersja skryptu znajduje się pod adresem https://bookdown.org/nowosad/Geostatystyka/.

Jeżeli używasz skryptu, zacytuj go jako:

- Nowosad, J., (2019). Geostatystyka w R. Poznań: Space A. ISBN 978-83-953296-0-9.

Zachęcam do zgłaszania wszelkich uwag, błędów, pomysłów oraz komentarzy na stronie https://github.com/Nowosad/geostat_book/issues.

[1] http://pandoc.org/

# Wymagania wstępne

## Oprogramowanie

Do odtworzenia przykładów użytych w poniższym skrypcie wystarczy podstawowa znajomość R. Aby zainstalować R oraz RStudio można skorzystać z poniższych odnośników:

- R[2] - https://cloud.r-project.org/
- RStudio[3] - https://www.rstudio.com/products/rstudio/download/

Dodatkowo, użyte zostały poniższe pakiety R (Kuhn, 2018; Hijmans et al., 2017; Nychka et al., 2019; Wickham et al., 2019; Auguie, 2017; Pebesma and Graeler, 2020; Appelhans et al., 2019; Giraudoux, 2018; Bivand et al., 2019; Bivand and Rundel, 2019; Pebesma and Bivand, 2019).

```
pakiety <- c(
  "caret", "dismo", "fields", "ggplot2", "gridExtra",
  "gstat", "mapview", "pgirmess", "rgdal", "rgeos", "sp"
)
```

Pakiety R używane w tym skrypcie można zainstalować za pomocą metapakietu **geostatbook** (Nowosad, 2020):

```
install.packages("devtools")
devtools::install_github("nowosad/geostatbook@2")
```

Jest to również możliwe poprzez funkcję `install.packages()`:

```
install.packages(pakiety)
```

## Dane

Dane wykorzystywane w tym skrypcie można pobrać w postaci spakowanego archiwum (dla rozdziału 2) oraz korzystając z pakietu **geostatbook** (dla kolejnych rozdziałów).

- Archiwum zawierające dane do rozdziału drugiego[4]
- Dane do kolejnych rozdziałów są zawarte w pakiecie geostatbook:[5]

[2] https://www.r-project.org/
[3] https://www.rstudio.com/
[4] https://github.com/Nowosad/geostat_book/blob/master/dane-wyd2.zip?raw=true
[5] https://github.com/Nowosad/geostatbook

```
# install.packages("devtools")
devtools::install_github("nowosad/geostatbook@2")
```

Aby ułatwić korzystanie ze skryptu, rozdziały od 3 do 13 rozpoczynają się od wczytania wymaganych pakietów oraz zbiorów danych.

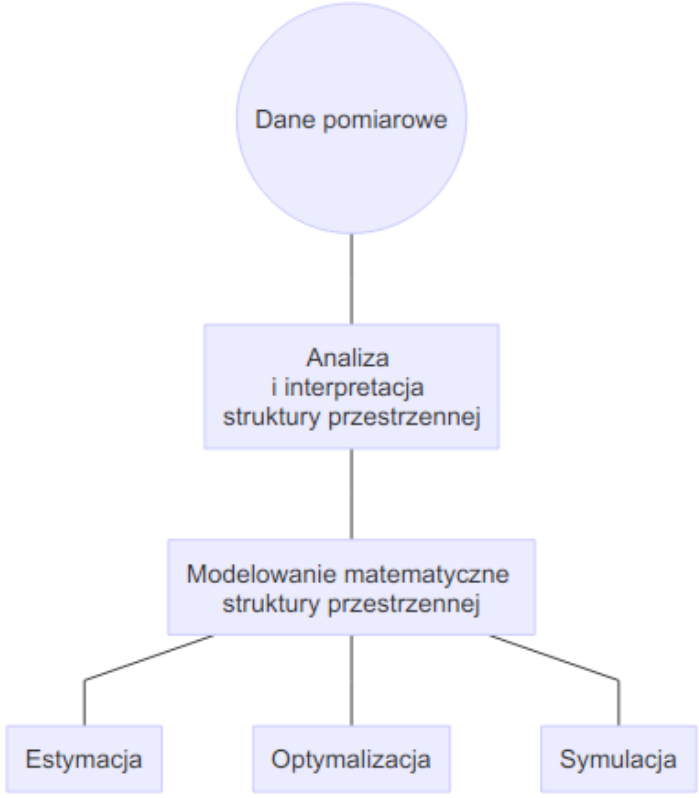
# 1. Wprowadzenie

## 1.1. Geostatystyczna analiza danych

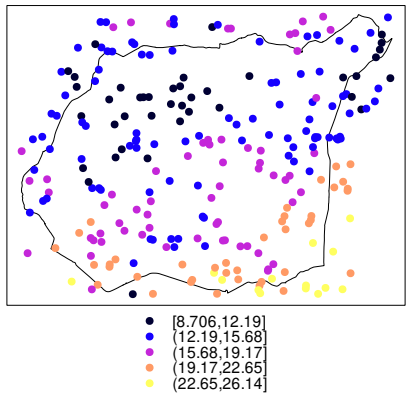
> Geostatystyka to gałąź statystyki skupiająca się na przestrzennych lub czasoprzestrzennych zbiorach danych

Geostatystyka jest stosowana obecnie w wielu dyscyplinach, jakich jak geologia naftowa, oceanografia, geochemia, logistyka, leśnictwo, gleboznawstwo, hydrologia, meteorologia, czy epidemiologia.

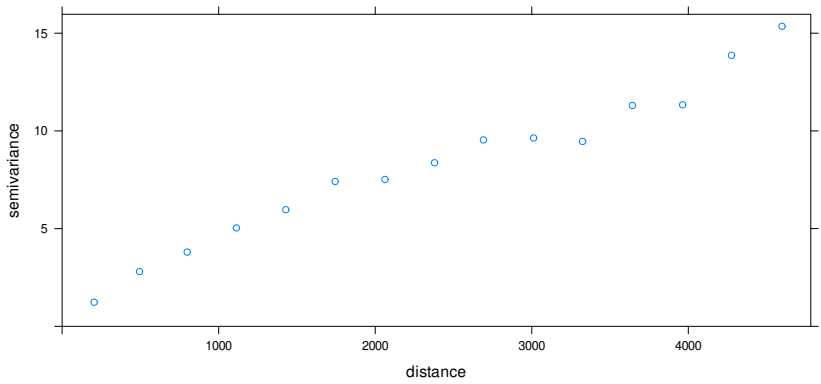
Geostatystyczna analiza danych może przyjmować różną postać w zależności od postawionego celu analizy. Poniższa rycina przestawia uproszczoną ścieżkę postępowania geostatystycznego.



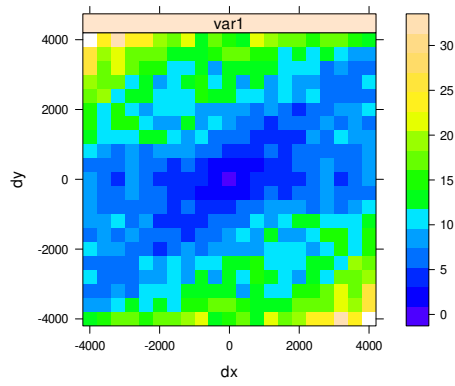

Punktem wyjścia analizy geostatystycznej jest posiadanie danych przestrzennych opisujących badane zjawisko, np. w **postaci punktowej**:



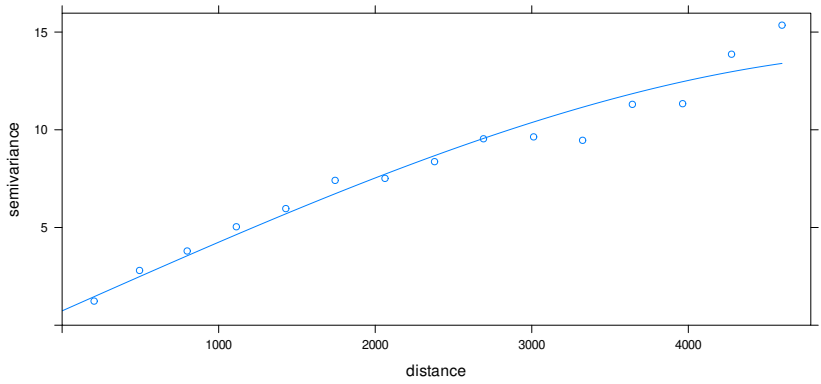

Na ich podstawie chcemy zrozumieć zmienność przestrzenną analizowanej cechy. Do tego może nam posłużyć wykres nazywany **semiwariogramem**:



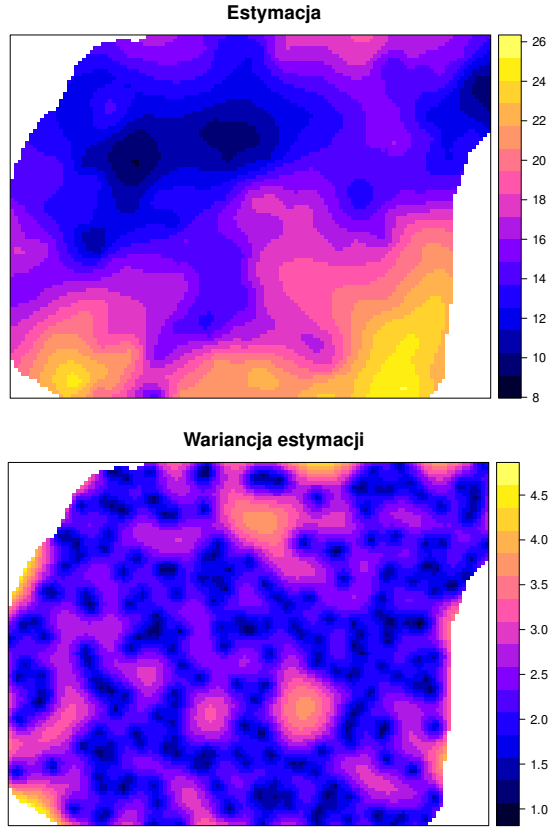

Opisuje on przestrzenną zmienność badanej cechy i za jego pomocą możemy stwierdzić jak szybko to zjawisko zmienia się w przestrzeni. Dodatkowo za pomocą **mapy semiwariogramu** możliwe jest stwierdzenie czy istnieją jakieś kierunki w których ta cecha zmienia się zmienia bardziej dynamicznie, a w których ta zmiana jest wolniejsza:



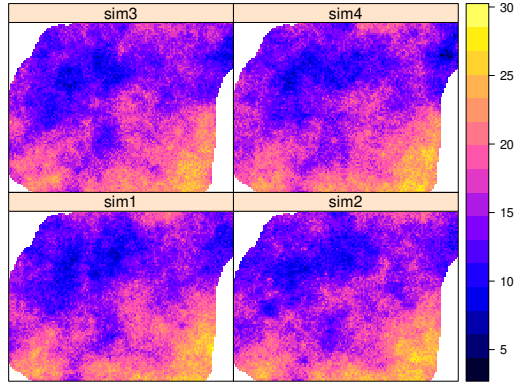

Następnie korzystając z wiedzy uzyskanej z semiwariogramu i mapy semiwariogramu, jesteśmy w stanie stworzyć **model semiwariogramu**:




Pozwala on zarówno na lepszy opis zmienności zjawiska, jak również służy do tworzenia **estymacji** czy też **symulacji**. Estymacja tworzy najbardziej potencjalnie możliwą wartość dla wybranej lokalizacji:




Rolą symulacji jest natomiast generowanie równie prawdopodobne możliwości rozkładu badanej cechy:



Każdy z powyższych elementów geostatystycznej analizy danych zostanie rozwinięty w dalszych rozdziałach tego skryptu.

# 2. R a dane przestrzenne

## 2.1. Wprowadzenie

### 2.1.1. Reprezentacja danych nieprzestrzennych

Skrypty i funkcje w języku R są zbudowane na podstawie szeregu obiektów nieprzestrzennych.

- Wektory (ang. *vector*):
  - liczbowe (ang. *integer*, *numeric*) - `c(1, 2, 3)` i `c(1.21, 3.32, 4.43)`
  - znakowe (ang. *character*) - `c("jeden", "dwa", "trzy")`
  - logiczne (ang. *logical*) - `c(TRUE, FALSE)`
  - czynnikowe (ang. *factor*) - `c("jeden", "dwa", "trzy", "jeden")`
- Ramki danych (ang. *data.frame*) - to zbiór zmiennych (kolumn) oraz obserwacji (wierszy) zawierających różne typy danych
- Macierze (ang. *matrix*)
- Listy (ang. *list*)

### 2.1.2. Pakiety

R zawiera także wiele funkcji pozwalających na przetwarzanie, wizualizację i analizowanie danych przestrzennych. Zawarte są one w szeregu pakietów (zbiorów funkcji), między innymi:

- GIS - **sp**, **rgdal**, **rgeos**, **sf**, **raster**, **stars**
- Geostatystyka - **gstat**, **geoR**, **geoRglm**

Więcej szczegółów na temat pakietów R służących do analizy przestrzennej można znaleźć pod adresem https://cran.r-project.org/web/views/Spatial.html.

### 2.1.3. Reprezentacja danych przestrzennych

Dane przestrzenne mogą być reprezentowane poprzez szereg różnych klas obiektów z użyciem różnych pakietów R. Przykładowo dane wektorowe

mogą być reprezentowane poprzez obiekty klas `Spatial*` z pakietu **sp** oraz obiekty klasy `sf` z pakietu **sf**.

Wszystkie obiekty klasy `Spatial*` z pakietu **sp** zawierają tablę atrybutów oraz dwie dodatkowe informacje:

```
- bounding box (`bbox`) - obwiednia - określa zasięg danych
- CRS (`proj4string`) - układ współrzędnych
```

Ten pakiet definiuje klasę obiektów - sposób reprezentacji danych. Najczęściej stosowane obiekty klasy `Spatial*` to `SpatialPointsDataFrame`, `SpatialPolygonsDataFrame` oraz `SpatialGridDataFrame`. Ostatnia klasa reprezentuje dane rastrowe.
Dodatkowo ten pakiet współpracuje z pakietami **rgdal** służącym do wczytywania i zapisywania danych oraz **rgeos** służącym do przetwarzania danych przestrzennych. W oparciu o pakiet **sp** powstało kilkaset dodatkowych pakietów R do analizy danych przestrzennych.

Dane rastrowe są reprezentowane między innymi poprzez klasę `SpatialGridDataFrame` z pakietu **sp** oraz obiekty klasy `Raster*` z pakietu **raster**, tj. `RasterLayer`, `RasterStack`, `RasterBrick`.

Więcej o pakiecie **sf** oraz **raster** można przeczytać w rozdziale drugim[1] książki Geocomputation with R.

### 2.1.4. GDAL/OGR

- Pakiety **rgdal**, **raster** czy **sf** wykorzystują biblioteki GDAL/OGR[2] w R do wczytywania i zapisywania danych przestrzennych
- GDAL to biblioteka zawierająca funkcje służące do odczytywania i zapisywania danych w formatach rastrowych
- OGR to biblioteka służąca to odczytywania i zapisywania danych w formatach wektorowych

### 2.1.5. PROJ

- Pakiety **sp**, **raster**, czy **sf** używają biblioteki PROJ[3] do określania i konwersji układów współrzędnych
- Dane przestrzenne powinny być zawsze powiązane z układem współrzędnych

[1] https://geocompr.robinlovelace.net/spatial-class.html
[2] http://www.gdal.org/
[3] https://proj4.org/

- PROJ to biblioteka pozwalająca na identyfikację oraz konwersję pomiędzy różnymi układami współrzędnych
- Strona http://www.spatialreference.org/ zawiera bazę danych układów współrzędnych
- Układy współrzędnych mogą być opisywane na szereg sposobów, np poprzez:
  - Reprezentację proj4string - określa ona szereg własności układu współrzędnych
  - Kod EPSG (ang. *European Petroleum Survey Group*) - pozwala on na łatwe identyfikowanie układów współrzędnych.
- Przykładowo, układ PL 1992 może być określony jako:

```
"+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000 +el-
lps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

- ...lub też za pomocą kodu EPSG:

```
"+init=epsg:2180"
```

### 2.1.6. Układ geograficzny

- Proporcje pomiędzy współrzędną oznaczającą długość geograficzną (X) a oznaczającą szerokość geograficzną (Y) nie są równe 1:1
- Jednostka mapy jest abstrakcyjna
- Wielkość oczka siatki jest zmienna
- Nie pozwala to na proste określanie odległości czy powierzchni
- Powyższe cechy układów geograficznych powodują, że do większości algorytmów w geostatystyce wykorzystywane są układy współrzędnych prostokątnych płaskich

### 2.1.7. Układ współrzędnych prostokątnych płaskich

- Określane są w miarach liniowych (np. metrach)
- Proporcje między współrzędną X a Y są równe 1:1
- Wielkość oczka jest stała

### 2.1.8. GEOS

- Pakiety **rgeos** czy **sf** używają biblioteki GEOS[4] do wykonywania operacji przestrzennych

[4] http://geos.osgeo.org/

- GEOS to biblioteka pozwalająca na przetwarzanie obiektów przestrzennych
- Przykładowe funkcje tej biblioteki to tworzenie buforów, wyliczanie centroidów, określanie relacji topologicznych (np. przecina, zawiera, etc.) i wiele innych

## 2.2. Import danych

R pozwala na odczytywanie danych przestrzennych z wielu formatów. Do najpopularniejszych należą dane tekstowe z plików .csv, dane wektorowe z plików .shp, dane rastrowe z plików w formacie GeoTIFF, oraz bazy danych przestrzennych z plików .gpkg.

### 2.2.1. Format .csv (dane punktowe)

Dane z plików tekstowych (np. .csv) można odczytać za pomocą uogólnionej funkcji `read.table()` lub też funkcji szczegółowych - `read.csv()` lub `read.csv2()`.

```
punkty_sp <- read.csv("dane/punkty.csv")
```

```
head(punkty_sp)
```

```
##       srtm clc      temp      ndvi      savi        x        y
## 1 175.7430   1 13.852222 0.6158061 0.4189449 750298.0 716731.6
## 2 149.8111   1 15.484209 0.5558816 0.3794864 753482.9 717331.4
## 3 272.8583  NA 12.760814 0.6067462 0.3745572 747242.5 720589.0
## 4 187.2777   1 14.324648 0.3756170 0.2386246 755798.9 718828.1
## 5 260.1366   1 15.908549 0.4598393 0.3087599 746963.5 717533.5
## 6 160.1416   2  9.941118 0.5600288 0.3453627 756801.6 720474.1
```

Po wczytaniu za pomocą funkcji `read.csv()`, nowy obiekt (np. `punkty_sp`) jest reprezentowany za pomocą klasy nieprzestrzennej `data.frame`. Aby obiekt został przetworzony do klasy przestrzennej, konieczne jest nadanie mu współrzędnych. W tym wypadku współrzędne znajdowały się w kolumnach `x` oraz `y`. Nadanie układu współrzędnych odbywa się poprzez funkcję `coordinates()`.

```
library(sp)
coordinates(punkty_sp) <- ~x + y
summary(punkty_sp)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## x 745592.5 756967.8
## y 712642.4 721238.7
## Is projected: NA
## proj4string : [NA]
## Number of points: 248
## Data attributes:
##       srtm            clc             temp             ndvi
##  Min.   :146.5   Min.   :1.000   Min.   : 7.883   Min.   :0.2024
##  1st Qu.:191.5   1st Qu.:1.000   1st Qu.:12.003   1st Qu.:0.4636
##  Median :217.9   Median :1.000   Median :14.941   Median :0.5154
##  Mean   :214.9   Mean   :1.481   Mean   :15.273   Mean   :0.5047
##  3rd Qu.:239.5   3rd Qu.:2.000   3rd Qu.:17.630   3rd Qu.:0.5742
##  Max.   :278.4   Max.   :4.000   Max.   :24.945   Max.   :0.6597
##  NA's   :3       NA's   :5       NA's   :1        NA's   :1
##       savi
##  Min.   :0.0824
##  1st Qu.:0.2935
##  Median :0.3256
##  Mean   :0.3176
##  3rd Qu.:0.3594
##  Max.   :0.4404
##  NA's   :1
```

Ważne, ale nie wymagane, jest także dodanie informacji o układzie przestrzennym danych za pomocą funkcji `proj4string()`.

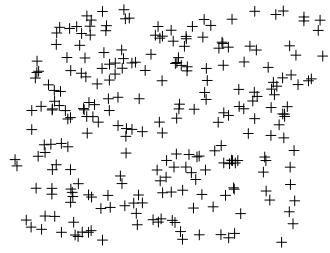
```
proj4string(punkty_sp) <- "+init=epsg:2180"
summary(punkty_sp)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## x 745592.5 756967.8
## y 712642.4 721238.7
## Is projected: TRUE
## proj4string :
## [+init=epsg:2180 +proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000
## +y_0=-5300000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 248
## Data attributes:
##       srtm            clc             temp             ndvi
##  Min.   :146.5   Min.   :1.000   Min.   : 7.883   Min.   :0.2024
```

```
##  1st Qu.:191.5   1st Qu.:1.000   1st Qu.:12.003   1st Qu.:0.4636
##  Median :217.9   Median :1.000   Median :14.941   Median :0.5154
##  Mean   :214.9   Mean   :1.481   Mean   :15.273   Mean   :0.5047
##  3rd Qu.:239.5   3rd Qu.:2.000   3rd Qu.:17.630   3rd Qu.:0.5742
##  Max.   :278.4   Max.   :4.000   Max.   :24.945   Max.   :0.6597
##  NA's   :3       NA's   :5       NA's   :1        NA's   :1
##       savi
##  Min.   :0.0824
##  1st Qu.:0.2935
##  Median :0.3256
##  Mean   :0.3176
##  3rd Qu.:0.3594
##  Max.   :0.4404
##  NA's   :1
```

Proste wyświetlenie uzyskanych danych klasy przestrzennej, np. `SpatialPointsDataFrame`, można uzyskać za pomocą funkcji `plot()`.

```
plot(punkty_sp)
```



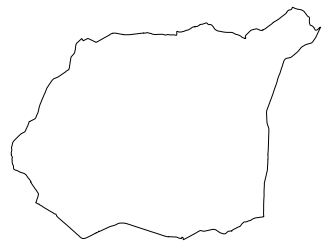
## 2.2.2. Dane poligonowe (formaty gisowe)

Dane wektorowe (np. GeoPackage czy ESRI Shapefile) można odczytać za pomocą funkcji `readOGR()` z pakietu **rgdal**. W najprostszej sytuacji przyjmuje ona jeden argument `dsn`, który oczekuje ścieżki do pliku wektorowego.

```
library(rgdal)
granica_sp <- readOGR(dsn = "dane/granica.gpkg")
```

```
## OGR data source with driver: GPKG
## Source: "/home/jn/Tmp/geostat_book-2/dane/granica.gpkg", layer: "granica.gpkg"
## with 1 features
## It has 3 fields
```

```
plot(granica_sp)
```



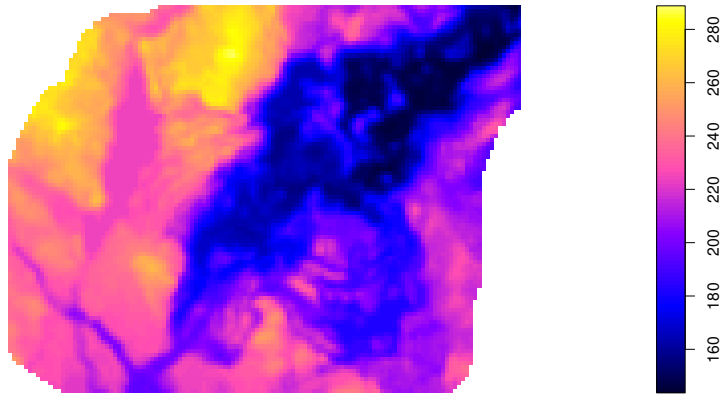
### 2.2.3. Rastry

Istnieje kilka sposobów odczytu danych rastrowych w R. Do najpopularniejszych należą funkcje `readGDAL()` z pakietu **rgdal** oraz `raster()` z pakietu **raster**. Należy w nich jedynie podać ścieżkę do pliku rastrowego.

```
siatka_sp <- readGDAL("dane/siatka.tif")
```

```
## dane/siatka.tif has GDAL driver GTiff
## and has 96 rows and 127 columns
```

```
plot(siatka_sp)
```




## 2.3. Przeglądanie danych przestrzennych

### 2.3.1. Struktura obiektu

Podstawowe informacje o obiektach można uzyskać za pomocą funkcji `str()`:

```
str(punkty_sp)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  248 obs. of  5 variables:
##   .. ..$ srtm: num [1:248] 176 150 273 187 260 ...
##   .. ..$ clc : int [1:248] 1 1 NA 1 1 2 2 1 1 2 ...
##   .. ..$ temp: num [1:248] 13.9 15.5 12.8 14.3 15.9 ...
##   .. ..$ ndvi: num [1:248] 0.616 0.556 0.607 0.376 0.46 ...
##   .. ..$ savi: num [1:248] 0.419 0.379 0.375 0.239 0.309 ...
##   ..@ coords.nrs : int [1:2] 6 7
##   ..@ coords     : num [1:248, 1:2] 750298 753483 747243 755799 746964 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:248] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "x" "y"
##   ..@ bbox       : num [1:2, 1:2] 745593 712642 756968 721239
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "x" "y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##  .. .. ..@ projargs: chr "+init=epsg:2180 +proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=50
5300000 +ellps=GRS80 +towgs84=0,0,0,"| __truncated__
```

Obiekty klas `Spatial*` mają pięć elementów (ang. *slots*) rozpoczynających się od symbolu `@`:

- `@data` - tabela atrybutów
- `@coords.nrs` - numer kolumn zawierających współrzędne
- `@coords` - współrzędne kolejnych elementów
- `@bbox` - obwiednia
- `@proj4string` - definicja układu współrzędnych

### 2.3.2. Tabla atrybutów

`@data` jest obiektem klasy `data.frame` zawierającym informacje o kolejnych punktach w tym zbiorze.

```
df <- punkty_sp@data
head(df)
```

```
##       srtm clc      temp      ndvi      savi
## 1 175.7430   1 13.852222 0.6158061 0.4189449
## 2 149.8111   1 15.484209 0.5558816 0.3794864
## 3 272.8583  NA 12.760814 0.6067462 0.3745572
## 4 187.2777   1 14.324648 0.3756170 0.2386246
## 5 260.1366   1 15.908549 0.4598393 0.3087599
## 6 160.1416   2  9.941118 0.5600288 0.3453627
```

### 2.3.3. Współrzędne

Element `@coords.nrs` określa numer kolumn zawierających współrzędne w oryginalnym zbiorze danych.

```
punkty_sp@coords.nrs
```

```
## [1] 6 7
```

Kolejny element, `@coords` definiuje współrzędne kolejnych elementów zadanego obiektu.

```
xy <- punkty_sp@coords
head(xy)
```

```
##          x        y
## 1 750298.0 716731.6
## 2 753482.9 717331.4
```

```
## 3 747242.5 720589.0
## 4 755798.9 718828.1
## 5 746963.5 717533.5
## 6 756801.6 720474.1
```

### 2.3.4. Obwiednia

Element `@bbox` określa zasięg przestrzenny danych w jednostkach mapy.

```
punkty_sp@bbox
```

```
##        min      max
## x 745592.5 756967.8
## y 712642.4 721238.7
```

### 2.3.5. Układ współrzędnych

`@proj4string` reprezentuje definicję układu współrzędnych.

```
punkty_sp@proj4string
```

```
## CRS arguments:
##  +init=epsg:2180 +proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000
## +y_0=-5300000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
```

## 2.4. Eksport danych

### 2.4.1. Zapisywanie danych wektorowych

R pozwala również na zapisywanie danych przestrzennych. W przypadku zapisu danych wektorowych za pomocą funkcji `writeOGR()` konieczne jest podanie nazwy zapisywanego obiektu (np. `granica`), folderu w którym chcemy zapisać plik (np. `dane`), nazwę zapisywanych plików bez rozszerzenia (np. `nowa_granica`), oraz sterownik - w przypadku danych shapefile jest to `ESRI Shapefile`.

```
# formaty wektorowe
writeOGR(granica_sp, dsn = "dane", layer = "nowa_granica",
         driver = "ESRI Shapefile")
```

Zapisywanie w bazach danych przestrzennych odbywa się poprzez nazwy zapisywanego obiektu (np. `granica`), ścieżki do zapisywanego pliku (np. `dane/nowa_granica.gpkg`), nazwę dla nowo utworzonej warstwy (np. `granica`), oraz sterownik - w przypadku formatu GeoPackage jest to `GPKG`.

```
# bazy danych przestrzennych
writeOGR(granica_sp, dsn = "dane/nowa_granica.gpkg",
         layer = "granica", driver = "GPKG")
```

### 2.4.2. Zapisywanie danych rastrowych

Funkcja `writeGDAL()` pozwala na zapisywanie danych rastrowych. Wymaga ona podania dwóch argumentów - nazwy zapisywanego obiektu (np. `siatka_sp`) oraz ścieżki i nazwy nowego pliku wraz z rozszerzeniem (np. `dane/nowy_siatka.tif`).

```
writeGDAL(siatka_sp, fname = "dane/nowy_siatka.tif")
```

Możliwe jest także użycie szeregu dodatkowych argumentów, np.:

- `type` - określającego typ danych wyjściowych
- `options` - pozwalającego na użycie dodatkowych opcji sterownika, np. użycie kompresji danych

## 2.5. Wizualizacja danych 2D

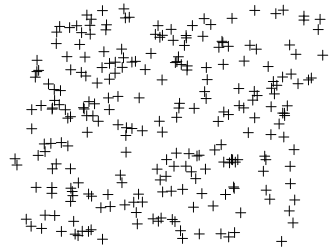
Do wizualizacji danych przestrzennych w R służy co najmniej kilkanaście różnych pakietów. Poniżej pokazane są przykłady kilku najprostszych funkcji - `plot()` oraz `spplot()` z pakietu `sp`. Więcej o wizualizacji danych przestrzennych można przeczytać w rozdziale Making maps with R[5] książki Geocomputation with R.

### 2.5.1. Dane punktowe

Funkcja `plot()` idealnie nadaje się do szybkiego przyjrzenia się, np. rodzajowi próbkowania danych.

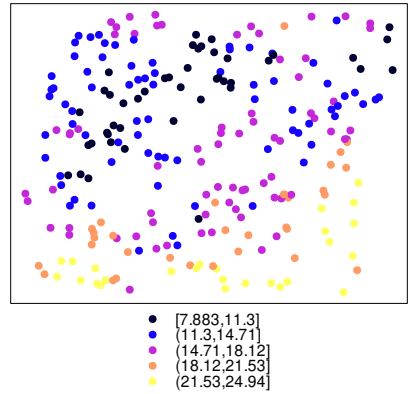
```
plot(punkty_sp)
```

[5] https://geocompr.robinlovelace.net/adv-map.html

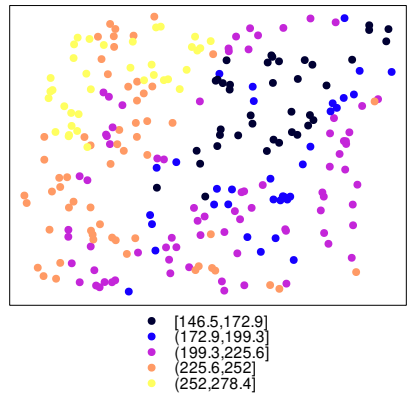


Funkcja `spplot()` w prosty sposób pozwala na obejrzenie rozkładu wartości wybranej zmiennej. Należy w niej podać nazwę obiektu oraz nazwę wyświetlanej zmiennej. Poniżej można zobaczyć przykłady dla zmiennych `temp` oraz `srtm`.

```
spplot(punkty_sp, "temp")
```



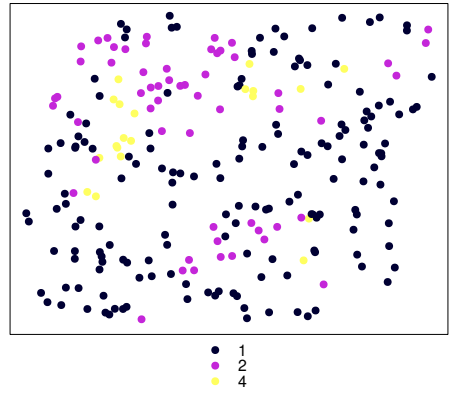

```
spplot(punkty_sp, "srtm")
```




### 2.5.2. Dane punktowe - kategorie

Nie zawsze dane mają postać ciągłych wartości - bywają one również określeniami różnych klas. W takich sytuacjach należy wcześniej przetworzyć typ danych do postaci kategorycznej (`as.factor()`). Następnie można je wyświetlić za pomocą funkcji `spplot()`.

```
punkty_sp@data$clc <- as.factor(punkty_sp@data$clc)
spplot(punkty_sp, "clc")
```

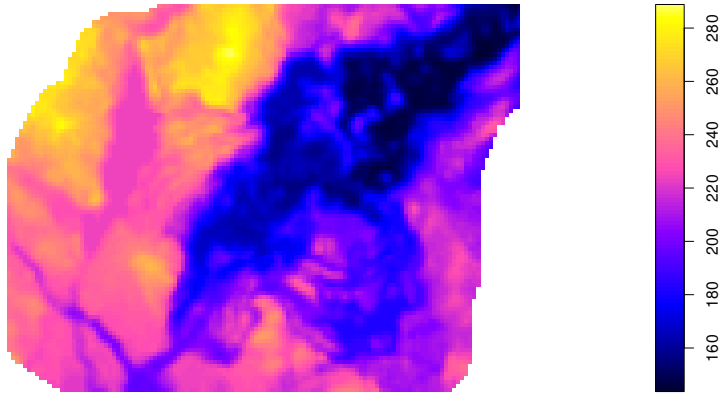


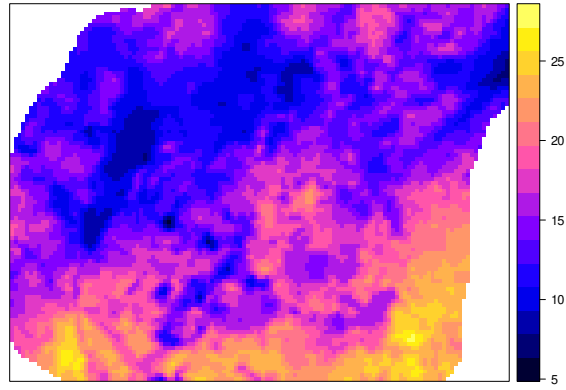

### 2.5.3. Rastry

Wyświetlanie danych w formacie rastrowym może odbyć się z użyciem funkcji `plot()` lub `spplot()`. Pierwsza z nich wymaga zdefiniowania obiektu

do wizualizacji i domyślnie wyświetla pierwszą warstwę obiektu. Druga z nich domyślnie wyświetla wszystkie dostępne warstwy lub też pojedynczą warstwę poprzez podanie jej nazwy:

```
# pierwsza zmienna
plot(siatka_sp)
# wybrana zmienna
spplot(siatka_sp, "band3")
```




## 2.6. Zadania

1. Wczytaj dane z pliku `punkty_ndvi.csv` i przetwórz je do postaci obiektu przestrzennego `ndvi_p`.
2. Stwórz mapę zmiennej `ndvi` z nowo utworzonego obiektu. (Dodatkowo: zapisz mapę do pliku graficznego `punkty_ndvi.png`).

3. Zapisz obiekt `ndvi_p` do formatu GeoPackage oraz do formatu ESRI Shapefile.

# 3. Eksploracyjna analiza danych nieprzestrzennych

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(ggplot2)
library(geostatbook)
data(punkty)
data(granica)
```

## 3.1. Cele eksploracyjnej analizy danych

Zazwyczaj przed przystąpieniem do analiz (geo-)statystycznych koniecznie jest wykonanie eksploracyjnej analizy danych nieprzestrzennych. Jej ogólne cele obejmują:

- Stworzenie ogólnej charakterystyki danych oraz badanego zjawiska.
- Określenie przestrzennego/czasowego typu próbkowania.
- Uzyskanie informacji o relacji pomiędzy lokalizacją obserwacji a czynnikami wpływającymi na zmienność przestrzenną badanych cech.

## 3.2. Dane

### 3.2.1. Struktura danych

Nie istnieje jedyna, optymalna ścieżka eksploracji danych. Proces ten różni się w zależności od posiadanego zbioru danych, jak i od postawionego pytania. Warto jednak, aby jednym z pierwszych kroków było przyjrzenie się danym wejściowym. Pozwala na to, między innymi, funkcja `str()`. Przykładowo, dla obiektu klasy `SpatialPointsDataFrame` wyświetla ona szereg ważnych informacji. Można tam odczytać, że obiekt `punkty` zawiera pięć elementów (ang. *slots*) rozpoczynających się od symbolu `@`. Pierwszy z

nich, `@data` jest obiektem klasy `data.frame` zawierającym informacje o kolejnych punktach w tym zbiorze. Element `@coords.nrs` określa numer kolumn zawierających współrzędne w oryginalnym zbiorze danych. W poniższym przypadku żadna kolumna nie ma takich informacji. Kolejny element, `@coords` definiuje współrzędne kolejnych punktów w obiekcie `punkty`. Ostatnie dwa elementy, `@bbox` i `@proj4string` określają kolejno zasięg przestrzenny danych oraz definicję ich układu współrzędnych.

```
str(punkty)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  250 obs. of  5 variables:
##   .. ..$ srtm: num [1:250] 231 185 269 212 209 ...
##   .. ..$ clc : num [1:250] 1 1 1 1 2 2 2 2 1 1 ...
##   .. ..$ temp: num [1:250] 22.5 16 16.1 17.1 22.3 ...
##   .. ..$ ndvi: num [1:250] 0.589 0.546 0.509 0.529 0.491 ...
##   .. ..$ savi: num [1:250] 0.357 0.382 0.342 0.345 0.22 ...
##   ..@ coords.nrs : num(0)
##   ..@ coords     : num [1:250, 1:2] 753638 749498 750131 751764 753676 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "x" "y"
##   ..@ bbox       : num [1:2, 1:2] 745547 712619 756937 721193
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "x" "y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##  .. .. ..@ projargs: chr "+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-
5300000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

Każdą ze zmiennych można obejrzeć oddzielnie, poprzez połączenie nazwy obiekty, znaku `$`, oraz nazwy elementu. Przykładowo `punkty$temp` pozwala na obejrzenie wartości zmiennej `temp` z tabeli atrybutów.

```
str(punkty$temp)
```

```
##  num [1:250] 22.5 16 16.1 17.1 22.3 ...
```

## 3.3. Statystyki opisowe

### 3.3.1. Podsumowanie numeryczne

Podstawową funkcją w R służącą wyliczaniu podstawowych statystyk jest `summary()`. Dla zmiennych numerycznych wyświetla ona wartość minimalną,

pierwszy kwartyl, medianę, średnią, trzeci kwartyl, oraz wartość maksymalną.

```
summary(punkty)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## x 745546.9 756937.4
## y 712618.8 721192.6
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 250
## Data attributes:
##       srtm            clc             temp             ndvi
##  Min.   :146.2   Min.   :1.000   Min.   : 8.706   Min.   :0.2102
##  1st Qu.:191.0   1st Qu.:1.000   1st Qu.:13.284   1st Qu.:0.4567
##  Median :218.2   Median :1.000   Median :15.309   Median :0.5037
##  Mean   :213.4   Mean   :1.268   Mean   :15.950   Mean   :0.5039
##  3rd Qu.:236.4   3rd Qu.:1.000   3rd Qu.:18.273   3rd Qu.:0.5521
##  Max.   :278.8   Max.   :4.000   Max.   :26.139   Max.   :0.6848
##       savi
##  Min.   :0.08343
##  1st Qu.:0.29017
##  Median :0.32212
##  Mean   :0.31847
##  3rd Qu.:0.35292
##  Max.   :0.48354
```

### 3.3.2. Średnia i mediana

Do określenia wartości przeciętnej zmiennych najczęściej stosuje się medianę i średnią.

```
median(punkty$temp, na.rm = TRUE)
```

```
## [1] 15.30944
```

```
mean(punkty$temp, na.rm = TRUE)
```

```
## [1] 15.95036
```

- W wypadku symetrycznego rozkładu te dwie cechy są równe
- Średnia jest bardziej wrażliwa na wartości odstające
- Mediana jest lepszą miarą środka danych, jeżeli są one skośne

Po co używać średniej?

- Bardziej przydatna w przypadku małych zbiorów danych
- Gdy rozkład danych jest symetryczny
- (Jednak) często warto podawać obie miary

### 3.3.3. Minimum i maksimum

Minimalna i maksymalna wartość zmiennej służy do określenia ekstremów w zbiorze danych, jak i sprawdzenia zakresu wartości.

```
min(punkty$temp, na.rm = TRUE)
```

```
## [1] 8.706485
```

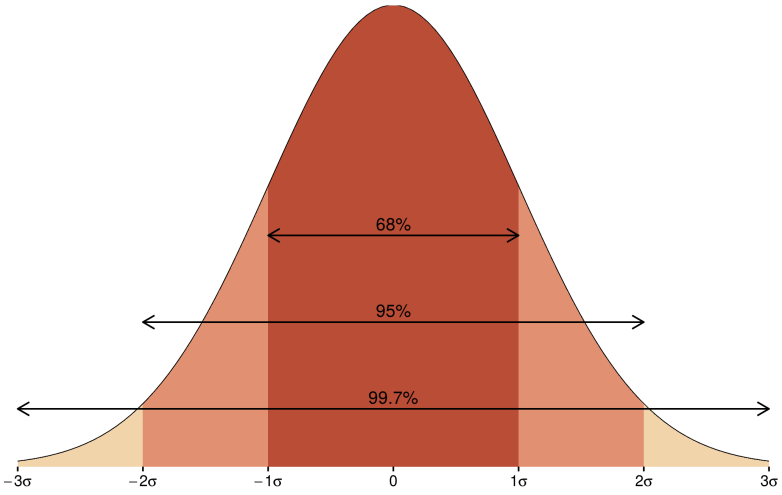
```
max(punkty$temp, na.rm = TRUE)
```

```
## [1] 26.13947
```

### 3.3.4. Odchylenie standardowe

Dodatkowo, często używaną statystyką jest odchylenie standardowe. Wartość ta określa w jak mocno wartości zmiennej odstają od średniej. Dla rozkładu normalnego ta wartość ma znane własności:

- 68% obserwacji mieści się w granicach jednego odchylenia standardowego od średniej
- 95% obserwacji mieści się w granicach dwóch odchyleń standardowych od średniej
- 99,7% obserwacji mieści się w granicach trzech odchyleń standardowych od średniej




```
sd(punkty$temp, na.rm = TRUE)
```

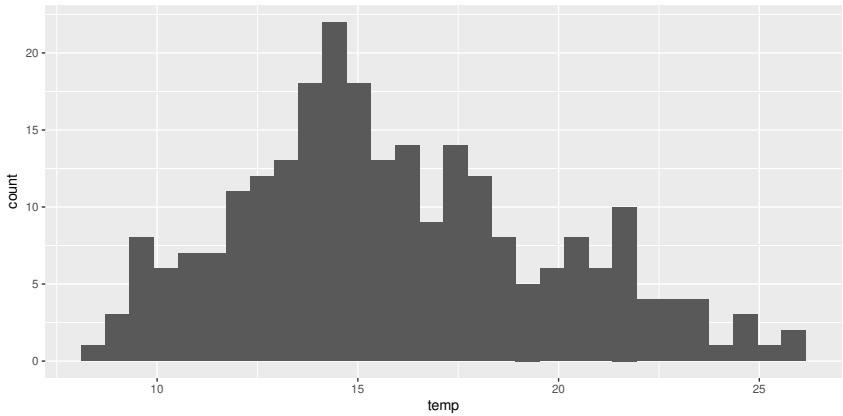
```
## [1] 3.839596
```

## 3.4. Wykresy

### 3.4.1. Histogram

Histogram należy do typów wykresów najczęściej używanych w eksploracyjnej analizie danych.

```
ggplot(punkty@data, aes(temp)) + geom_histogram()
```

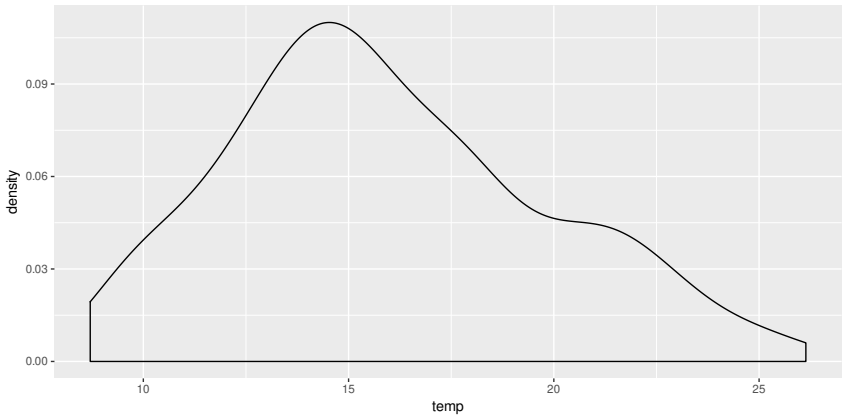


- Stworzony przez Karla Pearsona
- Jest graficzną reprezentacją rozkładu danych
- Wartości danych są łączone w przedziały (na osi poziomej) a na osi pionowej jest ukazana liczba punktów (obserwacji) w każdym przedziale
- Różny dobór przedziałów może dawać inną informację
- W pakiecie **ggplot2**, domyślnie przedział jest ustalany poprzez podzielenie zakresu wartości przez 30

### 3.4.2. Estymator jądrowy gęstości (ang. *kernel density estimation*)

Podobną funkcję do histogramu spełnia estymator jądrowy gęstości. Przypomina on wygładzony wykres histogramu i również służy graficznej reprezentacji rozkładu danych.

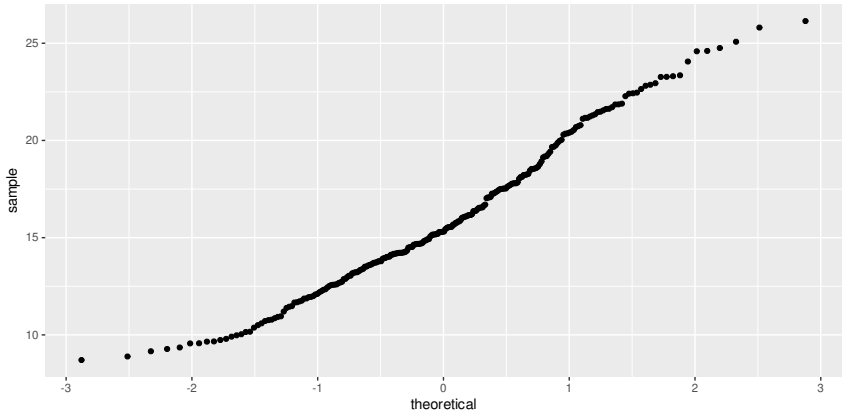
```
ggplot(punkty@data, aes(temp)) + geom_density()
```



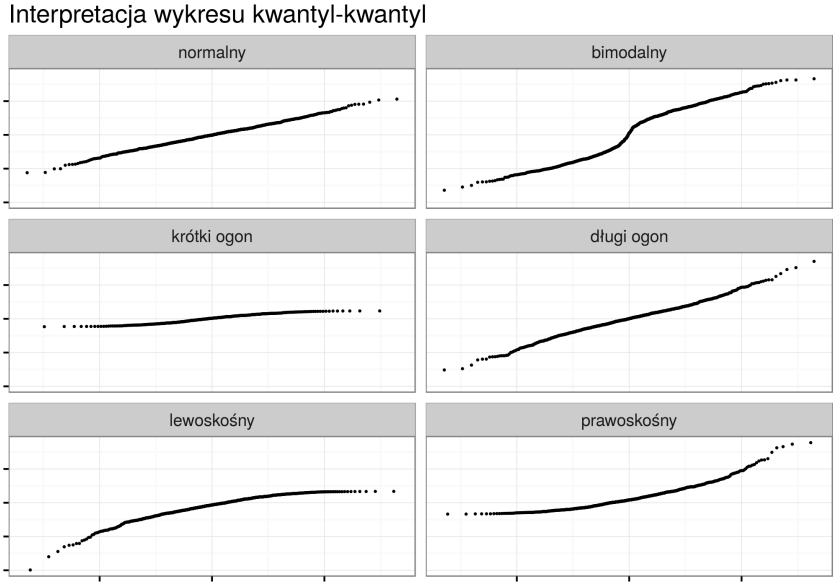

### 3.4.3. Wykres kwantyl-kwantyl (ang.*quantile-quantile*)

Wykres kwantyl-kwantyl ułatwia interpretację rozkładu danych.

```
ggplot(punkty@data, aes(sample = temp)) + geom_qq()
```



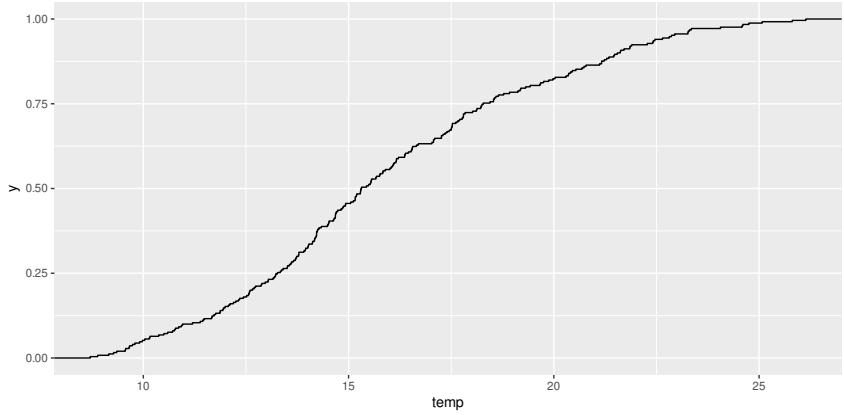

Na poniższej rycinie można zobaczyć przykłady najczęściej spotykanych cech rozkładu danych w wykresach kwantyl-kwantyl.




## 3.4.4. Dystrybuanta (CDF)

Dystrybuanta (ang. *cumulative distribution function* - CDF) wyświetla prawdopodobieństwo, że wartość zmiennej przewidywanej jest mniejsza lub równa określonej wartości.

```
ggplot(punkty@data, aes(temp)) + stat_ecdf()
```




## 3.5. Porównanie zmiennych

Wybór odpowiedniej metody porównania zmiennych zależy od szeregu cech, między innymi liczby zmiennych, ich typu, rozkładu wartości, etc.

### 3.5.1. Kowariancja

Kowariancja jest nieunormowaną miarą zależności liniowej pomiędzy dwiema zmiennymi. Kowariancja dwóch zmiennych $x$ i $y$ pokazuje jak dwie zmienne są ze sobą liniowo powiązane. Dodatnia kowariancja wskazuje na pozytywną relację liniową pomiędzy zmiennymi, podczas gdy ujemna kowariancja świadczy o odwrotnej sytuacji. Jeżeli zmienne nie są ze sobą liniowo powiązane, wartość kowariancji jest bliska zeru. Inaczej mówiąc, kowariancja stanowi miarę wspólnej zmienności dwóch zmiennych. Wielkość samej kowariancji uzależniona jest od przyjętej skali zmiennej (jednostki). Inne wyniku uzyskamy (przy tej samej zależności pomiędzy parą zmiennych), gdy będziemy analizować wyniki np. wysokości terenu w metrach i temperatury w stopniach Celsjusza a inne dla wysokości terenu w metrach i temperatury w stopniach Fahrenheita. Do wyliczania kowariancji w R służy funkcja `cov()`.

```
cov(punkty$temp, punkty$ndvi)
```

```
## [1] 0.02047509
```

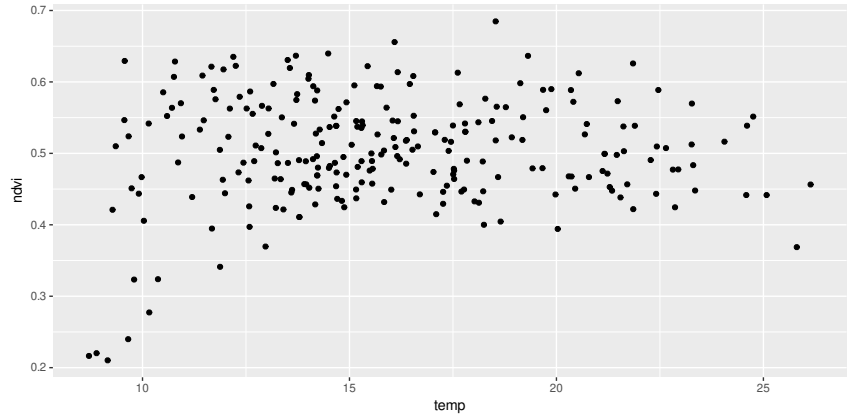
### 3.5.2. Współczynnik korelacji

Współczynnik korelacji to unormowana miara zależności pomiędzy dwiema zmiennymi, przyjmująca wartości od -1 do 1. Ten współczynnik jest uzyskiwany poprzez podzielenie wartości kowariancji przez odchylenie standardowe wyników. Z racji unormowania nie jest ona uzależniona od jednostki. Korelację można wyliczyć dzięki funkcji `cor()`.

```
cor(punkty$temp, punkty$ndvi)
```

```
## [1] 0.07049136
```

```
ggplot(punkty@data, aes(temp, ndvi)) + geom_point()
```




Dodatkowo funkcja `cor.test()` służy do testowania istotności korelacji. Za pomocą argumentu `method` można również wybrać jedną z trzech dostępnych miar korelacji.

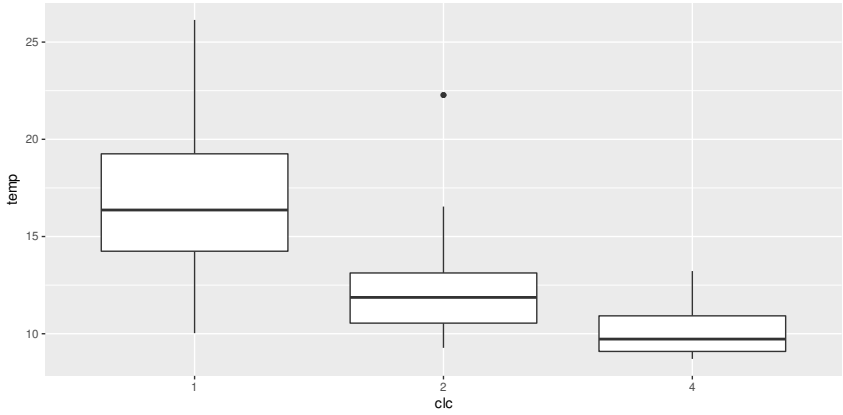
```
cor.test(punkty$temp, punkty$ndvi)
```

```
##
##  Pearson's product-moment correlation
##
## data:  punkty$temp and punkty$ndvi
## t = 1.1129, df = 248, p-value = 0.2668
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.05404834  0.19287158
## sample estimates:
##        cor
## 0.07049136
```

### 3.5.3. Wykres pudełkowy

Wykres pudełkowy obrazuje pięć podstawowych statystyk opisowych oraz wartości odstające. Pudełko to zakres międzykwartylowy (IQR), a linie oznaczają najbardziej ekstremalne wartości, ale nie odstające. Górna z nich to 1,5*IQR ponad krawędź pudełka, dolna to 1,5*IQR poniżej wartości dolnej krawędzi pudełka. Linia środkowa to mediana.

```
punkty$clc <- as.factor(punkty$clc)
ggplot(punkty@data, aes(x = clc, y = temp)) + geom_boxplot()
```




### 3.5.4. Testowanie istotności różnić średniej pomiędzy grupami

Analiza wariancji (ang. *Analysis of Variance* - ANOVA) służy do testowania istotności różnic między średnimi w wielu grupach. Metoda ta służy do oceny czy średnie wartości cechy $Y$ różnią się istotnie pomiędzy grupami wyznaczonymi przez zmienną $X$. ANOVA nie pozwala na stwierdzenie między którymi grupami występują różnice. Aby to stwierdzić konieczne jest wykonanie porównań wielokrotnych (*post-hoc*). ANOVĘ można wykonać za pomocą funkcji `aov()` definiując zmienną zależną oraz zmienną grupującą oraz zbiór danych.

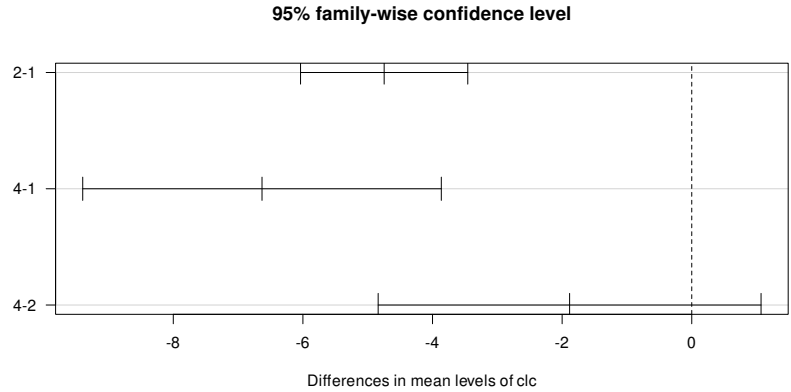
```
punkty$clc <- as.factor(punkty$clc)
aov_test <- aov(temp~clc, data = punkty)
summary(aov_test)
```

```
##              Df Sum Sq Mean Sq F value             Pr(>F)
```

```
## clc           2   1056   528.0   49.87 <0.0000000000000002 ***
## Residuals   247   2615    10.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Do wykonania porównań wielokrotnych służy funkcja `TukeyHSD()`. Dodatkowo wyniki można wizualizować za pomocą funkcji `plot()`.

```
tukey <- TukeyHSD(aov_test, "clc")
plot(tukey, las = 1)
```




# 3.6. Transformacje danych

## 3.6.1. Cel transformacji danych

Transformacja danych może mieć na celu ułatwienie porównywania różnych zmiennych, zniwelowanie skośności rozkładu lub też zmniejszenie wpływu danych odstających. W efekcie transformacja danych ułatwia przeprowadzenie analiz (geo-)statystycznych i polepsza wyniki prognoz z modeli.

## 3.6.2. Cechy standaryzacji

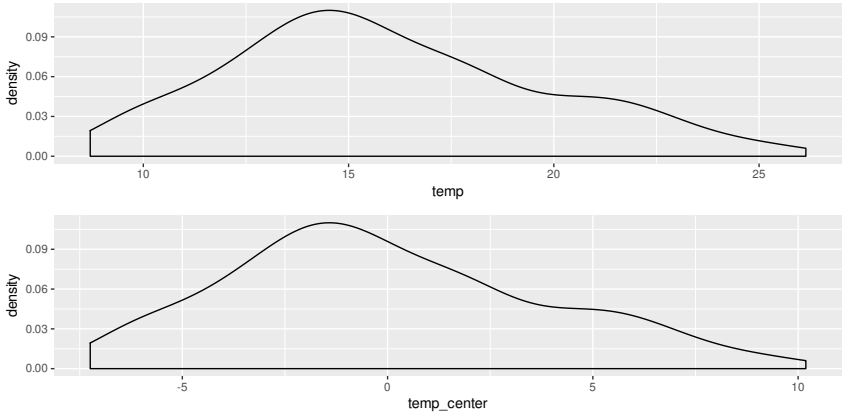
Centrowanie i skalowanie (standaryzacja):

- Centrowanie danych - wybierana jest przeciętna wartość predyktora, a następnie od wszystkich wartości predyktorów odejmowana jest wybrana wcześniej wartość

- Skalowanie danych - dzielenie każdej wartości predyktora przez jego odchylenie standardowe
- Wadą tego podejścia jest główne zmniejszenie interpretowalności pojedynczych wartości

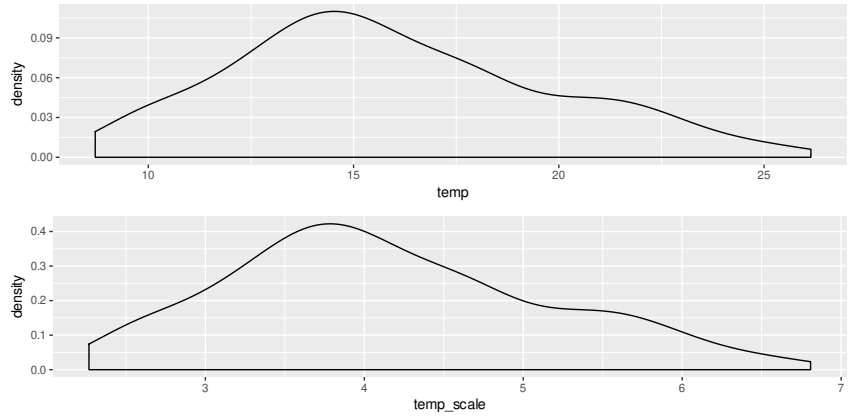
### 3.6.3. Centrowanie

```
ggplot(punkty@data, aes(temp)) + geom_density()
punkty$temp_center <- punkty$temp - mean(punkty$temp)
ggplot(punkty@data, aes(temp_center)) + geom_density()
```




### 3.6.4. Skalowanie

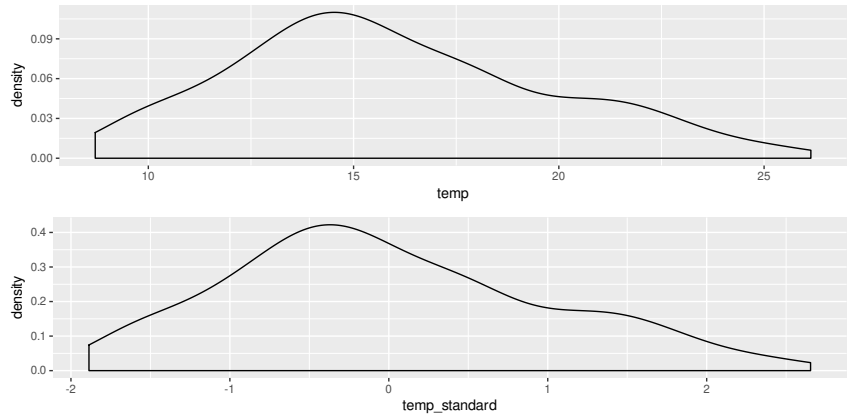
```
ggplot(punkty@data, aes(temp)) + geom_density()
punkty$temp_scale <- punkty$temp / sd(punkty$temp)
ggplot(punkty@data, aes(temp_scale)) + geom_density()
```




### 3.6.5. Standaryzacja

Do standaryzacji (centrowanie i skalowanie) służy funkcja `scale()`.

```
ggplot(punkty@data, aes(temp)) + geom_density()
punkty$temp_standard <- as.numeric(scale(punkty$temp))
ggplot(punkty@data, aes(temp_standard)) + geom_density()
```




### 3.6.6. Redukcja skośności

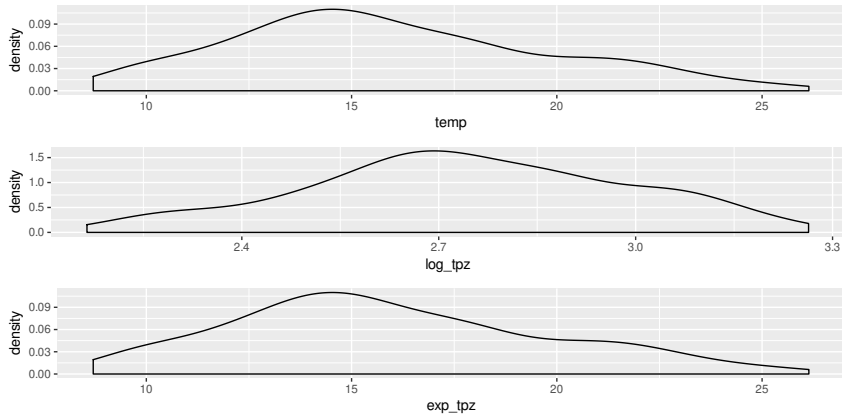
Redukcja skośności:

- Logarytmizacja
- Pierwiastkowanie
- Inne

### 3.6.7. Logarytmizacja

Logarytmizacja w R może odbyć się za pomocą funkcji `log()`. Transformację logarytmiczną można odwrócić używając funkcji `exp()`.

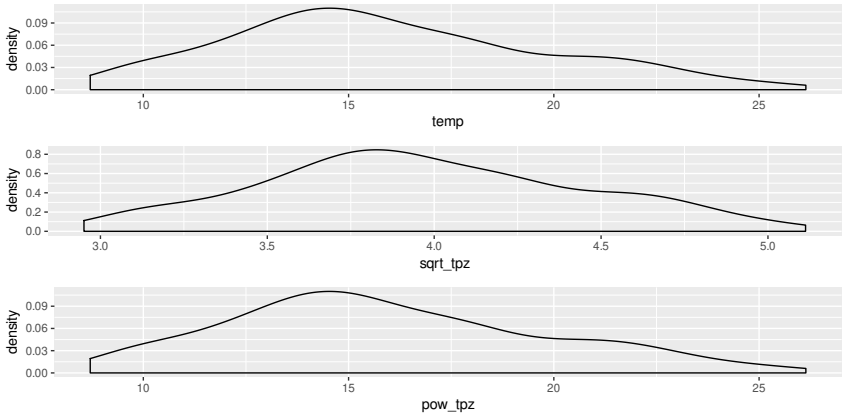
```
ggplot(punkty@data, aes(temp)) + geom_density()
punkty$log_tpz <- log(punkty$temp)
ggplot(punkty@data, aes(log_tpz)) + geom_density()
punkty$exp_tpz <- exp(punkty$log_tpz)
ggplot(punkty@data, aes(exp_tpz)) + geom_density()
```




### 3.6.8. Pierwiastkowanie

Pierwiastkowanie w R może odbyć się za pomocą funkcji `sqrt()`. Odwrócenie tej transformacji odbywa się poprzez podniesienie wartości do potęgi (`^2`).

```
ggplot(punkty@data, aes(temp)) + geom_density()
punkty$sqrt_tpz <- sqrt(punkty$temp)
ggplot(punkty@data, aes(sqrt_tpz)) + geom_density()
punkty$pow_tpz <- punkty$sqrt_tpz ^ 2
ggplot(punkty@data, aes(pow_tpz)) + geom_density()
```




## 3.7. Zadania

Przyjrzyj się danym z obiektu `punkty_pref`. Możesz go wczytać używając poniższego kodu:

```
data(punkty_pref)
```

1. Ile obserwacji i zmiennych jest zawartych w tym obiekcie? Jaka jest jego klasa?
2. Określ i opisz podstawowe statystyki opisowe zmiennej `ndvi`.
3. Stwórz histogram obrazujący rozkład tej zmiennej. Czym charakteryzuje się ten rozkład?
4. Jakie jest prawdopodobieństwo przekroczenia wartości `ndvi` 0.4 w tym zbiorze danych?
5. Określ korelację pomiędzy zmienną `ndvi` a `savi`. Czy jest ona istotna statystycznie?
6. Zbuduj wykres pokazujący zmienność `ndvi` w zależności od klasy pokrycia terenu. Czy istnieje istotna statystycznie różnica w wartościach `ndvi` względem klas pokrycia terenu?
7. Wypróbuj różne metody transformacji danych w celu redukcji skośności rozkładu zmiennej `ndvi`. Która metoda działa w tym przypadku najlepiej?
8. Stwórz powtarzalny przykład (więcej informacji w appendiksie C) używając pakiet **reprex** w oparciu o kod stworzony do rozwiązania poprzedniego zadania.

# 4. Eksploracyjna analiza danych przestrzennych

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(mapview)
library(geostatbook)
data(punkty)
data(granica)
```

## 4.1. Mapy

### 4.1.1. Podstawowe terminy | Kontekst przestrzenny

- Populacja - cały obszar, dla którego chcemy określić wybrane właściwości, np. temperatura powietrza.
- Próba - zbiór obserwacji, dla których mamy informacje, np. pomiary ze stacji meteorologicznych. Inaczej, próba to podzbiór populacji. Zazwyczaj niemożliwe (lub bardzo kosztowne) jest zdobycie informacji o całej populacji. Z tego powodu bardzo cenne jest odpowiednie wykorzystanie informacji z próby.

### 4.1.2. Mapy punktowe

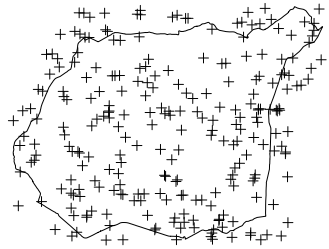
Eksploracyjna analiza danych przestrzennych w przypadku danych punktowych ma na celu:

- Sprawdzenie poprawności współrzędnych
- Wgląd w typ próbkowania danych
- Sprawdzenie poprawności danych, w tym między innymi określenie danych odstających lokalnie
- Identyfikacja głównych cech struktury przestrzennej zjawiska (np. trend)

## 4.2. Sprawdzenie poprawności współrzędnych

Wstępne sprawdzenie poprawności współrzędnych można wykonać poprzez wizualizację danych przestrzennych za pomocą funkcji `plot()`.

```
plot(granica)
plot(punkty, add = TRUE)
```



## 4.3. Próbkowanie

### 4.3.1. Podstawowe typy próbowania

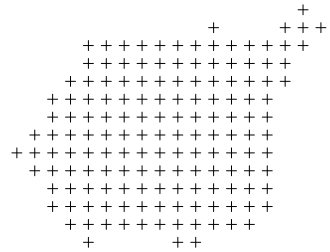
Istnieje cały szereg typów próbkowania danych przestrzennych. Funkcja `spsample()` z pakietu **sp** pozwala na stworzenie kilku typów próbkowania (argument `type`), między innymi:

- Regularny (ang.*regular*)
- Losowy (ang.*random*)
- Losowy stratyfikowany (ang.*stratified*)
- Preferencyjny (ang.*clustered*)

### 4.3.2. Typ próbowania | Regularny

W regularnym typie próbkowania, kolejne punkty rozłożone są równomiernie na badanym obszarze.

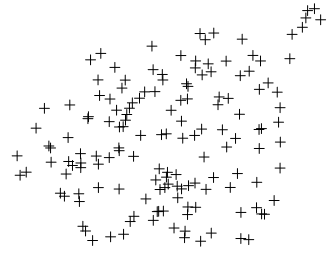
```
set.seed(225)
regularny <- spsample(granica, 150, type = "regular")
plot(regularny)
```



### 4.3.3. Typ próbowania | Losowy

W losowym typie próbkowania każda lokalizacja ma takie samo prawdopodobieństwo wystąpienia. Dodatkowo, każdy punkt jest losowany niezależnie od pozostałych.

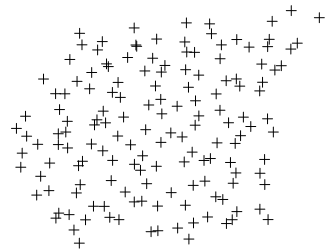
```
set.seed(301)
losowy <- spsample(granica, 150, type = "random")
plot(losowy)
```



### 4.3.4. Typ próbowania | Losowy stratyfikowany

Losowy stratyfikowany typ próbkowania polega na podzieleniu analizowanego obszaru na regularne komórki, a następnie dla każdej komórki losowana jest lokalizacja punktu.

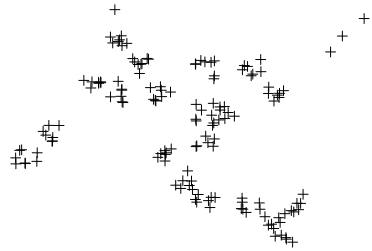
```
set.seed(125)
strat <- spsample(granica, 150, type = "stratified")
plot(strat)
```



### 4.3.5. Typ próbowania | Preferencyjny

W preferencyjnym typie próbkowania istnieją obszary, które z jakieś powodu (np. specyficzne wartości analizowanej cechy) są znacznie częściej opróbowane niż inne.

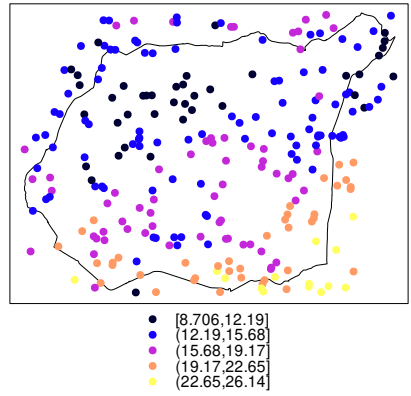
```
set.seed(425)
pref <- spsample(granica, 150, type = "clustered",
                 nclusters = 30)
plot(pref)
```



## 4.4. Dane lokalnie odstające

Dane lokalnie odstające oznaczają nietypowe przestrzennie wartości danej cechy. Inaczej mówiąc, może to być niska wartość otoczona wysokimi wartościami lub też wysoka wartość otoczona niskimi wartościami. Dane lokalnie ostające mogą oznaczać zarówno błąd w danych albo wpływ innego czynnika na analizowaną cechę. Przyjrzenie się wartościom analizowanej cechy można wykonać z użyciem funkcji `plot()`. Na poniższym przykładzie wyświetlona jest zmienna `temp` oznaczająca średnią temperaturę dobową.

```
spplot(punkty, "temp", sp.layout = granica)
```




Dodatkowo można wykorzystać pakiet **mapview** do interaktywnego określania wartości oraz numeru punktu (numer wiersza w tabeli atrybutów).

```
mapview(punkty["temp"])
```

## 4.5. Zadania

1. Określ i opisz typ próbkowania danych `punkty`.
2. Wykonaj mapę pokazującą przestrzenny rozkład wartości NDVI w zbiorze `punkty`. (Dodatkowo: spróbuj użyć do tego pakietu **tmap** - `install.packages("tmap")`).

# 5. Metody interpolacji

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(rgdal)
library(gstat)
library(dismo)
library(fields)
library(geostatbook)
data(punkty)
data(siatka)
data(granica)
```

Przez przejściem do interpolacji geostatystycznych warto zdać sobie sprawę, że nie jest to jedyna możliwa droga postępowania do stworzenia estymacji przestrzennych. Można wyróżnić dwie główne grupy modeli przestrzennych - modele deterministyczne oraz modele statystyczne.

## 5.1. Tworzenie siatek

W większości przypadków analiz geostatystycznych konieczne jest stworzenie siatki interpolacyjnej (pustego rastra). Istnieją dwa podstawowe rodzaje takich siatek - siatki regularne oraz siatki nieregularne.

### 5.1.1. Siatki regularne

Siatki regularne mają kształt prostokąta obejmującego cały analizowany obszar. Określenie granic obszaru można wykonać na podstawie zasięgu danych punktowych za pomocą funkcji `bbox()` z pakietu **sp**.

```
bbox(punkty)
```

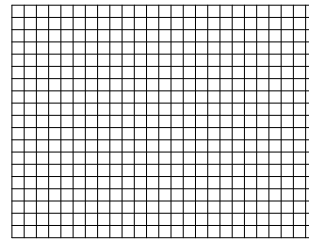
```
##        min      max
## x 745546.9 756937.4
## y 712618.8 721192.6
```

Do stworzenia siatki można użyć funkcji `expand.grid()`. Wymaga ona określenia dwóch argumentów - `x` oraz `y` (taka ich nazwa nie jest obowiązkowa). Oba argumenty przyjmują trzy wartości: (i) `from` - oznaczający wartość początkową współrzędnej, (ii) `to` - określający wartość końcową współrzędnej, oraz (iii) `by` - określający rozdzielczość. Przy ustalaniu wartości początkowej i końcowej konieczne jest ich rozszerzenie względem wartości z funkcji `bbox()`, aby wszystkie analizowane punkty znalazły się na badanym obszarze.

```
nowa_siatka <- expand.grid(
  x = seq(from = 745050, to = 757050, by = 500),
  y = seq(from = 712650, to = 721650, by = 500)
)
```

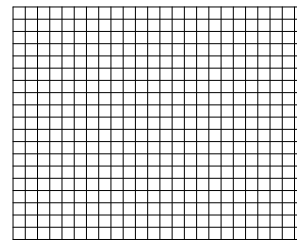
Utworzony w ten sposób obiekt wymaga określenia współrzędnych (funkcja `coordinates()`), potwierdzenia że dane mają być siatką (funkcja `gridded()`), oraz przypisania układu współrzędnych z obiektu punktowego (funkcja `proj4string()`).

```
coordinates(nowa_siatka) <- ~x + y
gridded(nowa_siatka) <- TRUE
proj4string(nowa_siatka) <- proj4string(punkty)
plot(nowa_siatka)
```



Alternatywnie, do stworzenia siatki można wykorzystać funkcję `makegrid()`. Tworzy ona nowy obiekt na podstawie istniejącego obiektu punktowego oraz zadanej rozdzielczości.

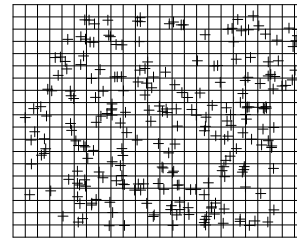
```
nowa_siatka <- makegrid(punkty, cellsize = 500)
coordinates(nowa_siatka) <- ~x1 + x2
gridded(nowa_siatka) <- TRUE
proj4string(nowa_siatka) <- proj4string(punkty)
plot(nowa_siatka)
```



### 5.1.2. Siatki regularne - wizualizacja

Sprawdzenie, czy uzyskana siatka oraz dane punktowe się na siebie nakładają można wykonać za pomocą funkcji `plot()`. W poniższym przykładzie, pierwszy wiersz służy wyświetleniu siatki, a drugi dodaje dane punktowe z użyciem argumentu `add`.

```
plot(nowa_siatka)
plot(punkty, add = TRUE)
```



### 5.1.3. Siatki nieregularne

Siatki nieregularne mają zazwyczaj kształt wieloboku obejmującego analizowany obszar. Mogą one powstać, np. w oparciu o wcześniej istniejące granice. Siatki nieregularne w R mogą być reprezentowane poprzez klasę `SpatialPixelsDataFrame`.

W poniższym przypadku odczytywana jest granica badanego obszaru z pliku GeoPackage. Taki obiekt można np. stworzyć za pomocą oprogramowania GISowego takiego jak QGIS[1]. Następnie tworzony jest nowy obiekt `nowa_siatka_n` poprzez wybranie tylko tych oczek siatki, które znajdują się wewnątrz zadanych granic.

```
granica <- readOGR("dane/granica.gpkg")
```

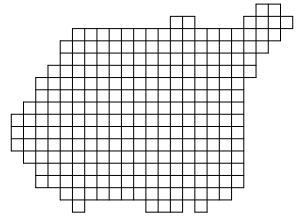
```
## OGR data source with driver: GPKG
## Source: "/home/jn/Tmp/geostat_book-2/dane/granica.gpkg", layer: "granica.gpkg"
## with 1 features
## It has 3 fields
```

```
nowa_siatka_n <- nowa_siatka[granica]
```

Wynik przetworzenia można zobaczyć z użyciem funkcji `plot`.

```
plot(nowa_siatka_n)
```

[1] http://www.qgis.org/pl/site/



## 5.2. Modele deterministyczne

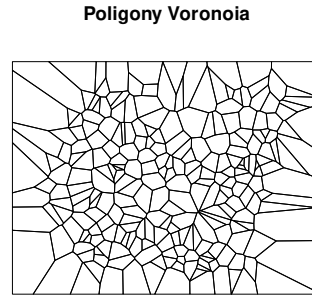
Modele deterministyczne charakteryzują się tym, że ich parametry są zazwyczaj ustalane w oparciu o funkcję odległości lub powierzchni. Zaletą tych modeli jest ich prosta oraz krótki czas obliczeń. W tych modelach brakuje jednak szacunków na temat oceny błędu modelu. Do modeli deterministycznych należą, między innymi:

- Metoda diagramów Voronoi'a (ang. *Voronoi diagram*)
- Metoda średniej ważonej odległością (ang. *Inverse Distance Weighted - IDW*)
- Funkcje wielomianowe (ang. *Polynomials*)
- Funkcje sklejane (ang. *Splines*)

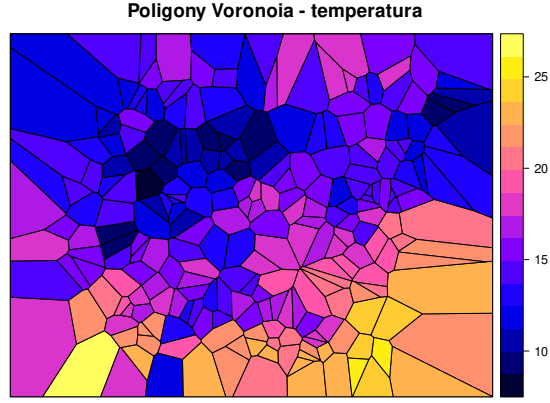
### 5.2.1. Voronoi

Metoda diagramów Voronoi'a polega na stworzeniu nieregularnych poligonów na podstawie analizowanych punktów, a następnie wpisaniu w każdy poligon wartości odpowiadającego punktu. Na poniższym przykładzie ta metoda stosowana jest z użyciem funkcji `voronoi()` z pakietu **dismo**.

```
voronoi_interp <- voronoi(punkty)
plot(voronoi_interp, main = "Poligony Voronoia")
```




```
spplot(voronoi_interp, zcol = "temp",
       main = "Poligony Voronoia - temperatura")
```



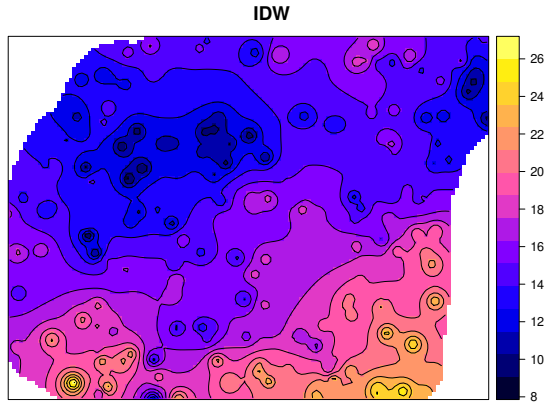

## 5.2.2. IDW

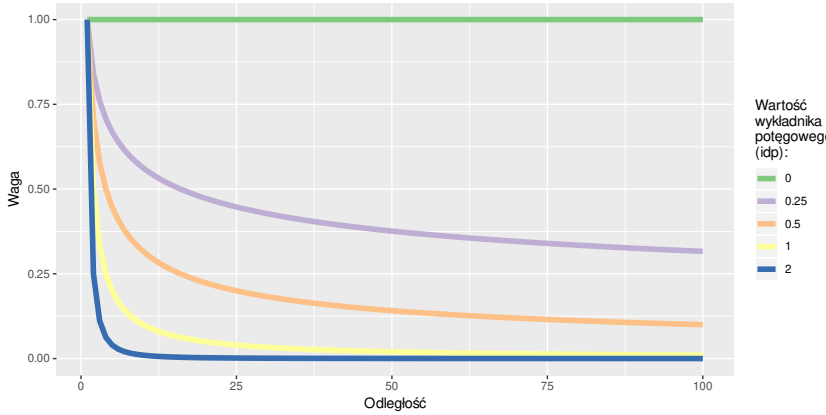
Metoda średniej ważonej odległością (IDW) wylicza wartość dla każdej komórki na podstawie wartości punktów obokległych ważonych odwrotnością ich odległości. W efekcie, czym bardziej jest punkt oddalony, tym mniejszy jest jego wpływ na interpolowaną wartość. Wagę punktów ustala się z użyciem argumentu wykładnika potęgowego (ang. *power*). W pakiecie **gstat** istnieje do tego celu funkcja `idw()`, która przyjmuje analizowaną cechę (`temp~1`), zbiór punktowy, siatkę, oraz wartość wykładnika potęgowego (argument `idp`).

```
idw_interp <- idw(temp~1, locations = punkty,
                  newdata = siatka, idp = 2)

## [inverse distance weighted interpolation]

spplot(idw_interp, zcol = "var1.pred", contour = TRUE,
       main = "IDW")
```




## 5.2.3. Funkcje wielomianowe

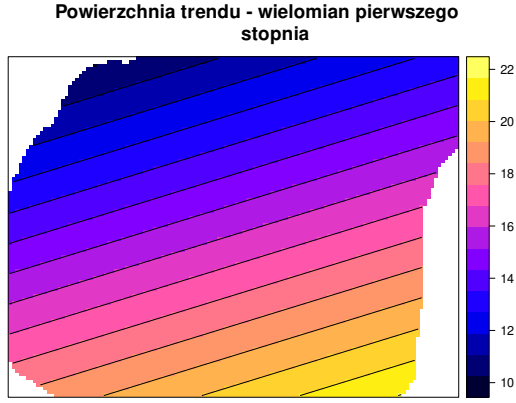
Stosowanie funkcji wielomianowych w R może odbyć się z wykorzystaniem funkcji `gstat()` z pakietu **gstat**. Wymaga ona podania trzech argumentów:

`formula` określającego naszą analizowaną cechę (`temp~1` mówi, że chcemy interpolować wartość temperatury zależnej od samej siebie), `data` określający analizowany zbiór danych, oraz `degree` określającą stopień wielomianu. Następnie funkcja `predict()` przenosi nowe wartości na wcześniej stworzoną siatkę.

```
# wielomian 1 stopnia
wielomian_1 <- gstat(formula = temp~1, locations = punkty,
                     degree = 1)
wielomian_1_pred <- predict(wielomian_1, newdata = siatka)
```

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

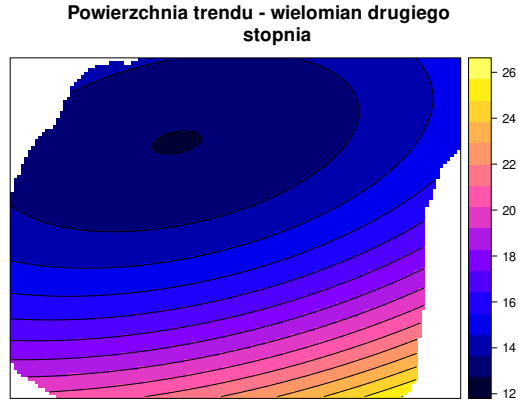
```
spplot(wielomian_1_pred, zcol = "var1.pred", contour = TRUE,
       main = "Powierzchnia trendu - wielomian pierwszego
       stopnia")
```




```
# wielomian 2 stopnia
wielomian_2 <- gstat(formula = temp~1, locations = punkty,
                     degree = 2)
wielomian_2_pred <- predict(wielomian_2, newdata = siatka)
```

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

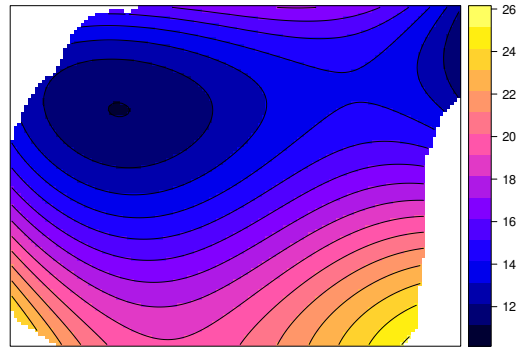
```
spplot(wielomian_2_pred, zcol = "var1.pred", contour = TRUE,
       main = "Powierzchnia trendu - wielomian drugiego
       stopnia")
```




```
# wielomian 3 stopnia
wielomian_3 <- gstat(formula = temp~1, locations = punkty,
                     degree = 3)
wielomian_3_pred <- predict(wielomian_3, newdata = siatka)
```

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

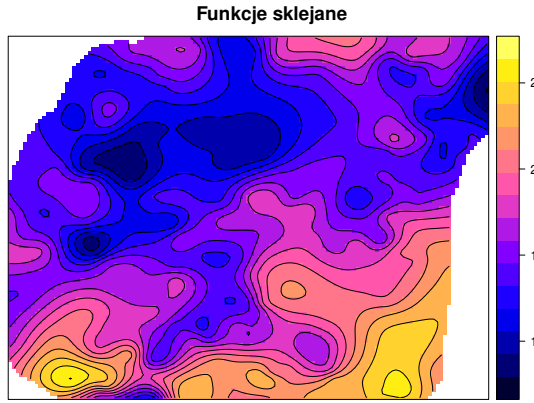
```
spplot(wielomian_3_pred, zcol = "var1.pred", contour = TRUE,
       main = "Powierzchnia trendu - wielomian trzeciego
       stopnia")
```



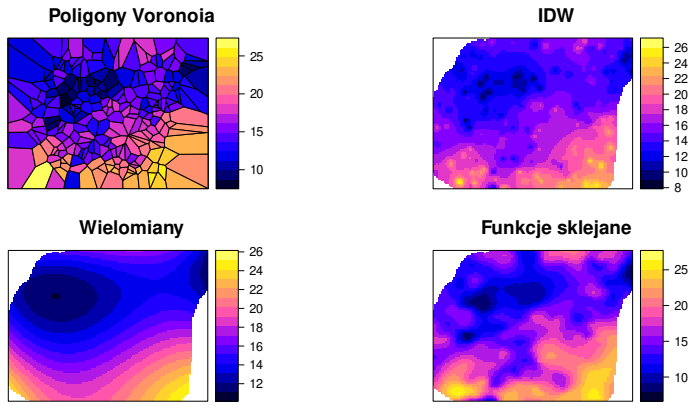
### 5.2.4. Funkcje sklejane

Interpolacja z użyciem funkcji sklejanych (funkcja `Tps()` z pakietu **fields**) dopasowuje krzywą powierzchnię do wartości analizowanych punktów.

```
tps <- Tps(coordinates(punkty), punkty$temp)
siatka$tps_pred <- predict(tps, coordinates(siatka))
spplot(siatka, "tps_pred", contour = TRUE,
       main = "Funkcje sklejane")
```




### 5.2.5. Porównanie modeli deterministycznych



## 5.3. Modele statystyczne

Modele statystyczne charakteryzują się tym, że ich parametry określane są w oparciu o teorię prawdopodobieństwa. Dodatkowo wynik estymacji zawiera także oszacowanie błędu, jednak te metody zazwyczaj wymagają większych zasobów sprzętowych. Do modeli statystycznych należą, między innymi:

- Kriging
- Modele regresyjne
- Modele bayesowskie
- Modele hybrydowe

W kolejnych rozdziałach można znaleźć omówienie kilku podstawowych typów pierwszej z tych metod - krigingu.

## 5.4. Zadania

1. Stwórz siatkę interpolacyjną o rozdzielczości 200 metrów dla obszaru Suwalskiego Parku Krajobrazowego.
2. Korzystając z danych `punkty` wykonaj interpolację zmiennej `srtm` używając:

- Poligonów Voronoi'a
- Metody IDW
- Funkcji wielomianowych
- Funkcji sklejanych

3. Porównaj uzyskane wyniki poprzez ich wizualizację. Czym różnią się powyższe metody?
4. Wykonaj interpolację zmiennej `temp` metodą IDW sprawdzając różne parametry argumentu `idp`. W jaki sposób wpływa on na uzyskaną interpolację?

# 6. Miary relacji przestrzennych

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(rgeos)
library(gstat)
library(pgirmess)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
data(granica)
```

## 6.1. Geostatystyka

### 6.1.1. Definicja

Geostatystyka jest to zbiór narzędzi statystycznych uwzględniających w analizie danych ich przestrzenną i czasową lokalizację, a opartych o teorię funkcji losowych.

### 6.1.2. Funkcje

Istnieją cztery główne funkcje geostatystyki:

1. Identyfikacja i modelowanie struktury przestrzennej/czasowej zjawiska
2. Estymacja - szacowanie wartości badanej zmiennej w nieopróbowanym miejscu i/lub momencie czasu
3. Symulacja - generowanie alternatywnych obrazów, które honorują wyniki pomiarów i strukturę przestrzenną/czasową zjawiska
4. Optymalizacja próbkowania/sieci pomiarowej

Inaczej mówiąc, celem geostatystyki nie musi być tylko interpolacja (estymacja) przestrzenna, ale również zrozumienie zmienności przestrzennej lub czasowej zjawiska, symulowanie wartości, oraz optymalizacja sieci pomiarowej.

### 6.1.3. Podstawowe etapy

W przypadku estymacji geostatystycznej, zwanej inaczej interpolacją geostatystyczną, pełna ścieżka postępowania składa się z siedmiu elementów:

1. Zaprojektowanie sposobu (typu) próbkowania oraz organizacji zadań
2. Zebranie danych, analiza laboratoryjna
3. Wstępna eksploracja danych, ocena ich jakości
4. Modelowanie semiwariogramów na podstawie dostępnych danych
5. Estymacja badanej cechy
6. Porównanie i ocena modeli
7. Stworzenie wynikowego produktu i jego dystrybucja

### 6.1.4. Dane wejściowe

Jedną z najważniejszych ograniczeń stosowania metod geostatystycznych jest spełnienie odpowiednich założeń dotyczących danych wejściowych. Muszą one:

1. Zawierać wystarczająco dużą liczbę punktów (minimalnie >30, ale zazwyczaj więcej niż 100/150)
2. Być reprezentatywne
3. Być niezależne
4. Być stworzone używając stałej metodyki
5. Być wystarczająco dokładne

### 6.1.5. Badane zjawisko

Oprócz ograniczeń dotyczących danych wejściowych, istnieją również założenia dotyczące analizowanej cechy (analizowanego zjawiska):

1. Przestrzennej ciągłości - przestrzenna korelacja między zmienny w dwóch lokalizacjach zależy tylko od ich odległości (oraz czasem kierunku), lecz nie od tego gdzie są one położone
2. Stacjonarności - średnia i wariancja są stałe na całym badanym obszarze

### 6.1.6. Podstawowe symbole

Podstawowe symbole w określaniu przestrzennej zmienności to:

- $s$ - wektor współrzędnych
- $z(s)$ - badana zmienna jako funkcja położenia - inaczej określany jako ogon (ang. *tail*)

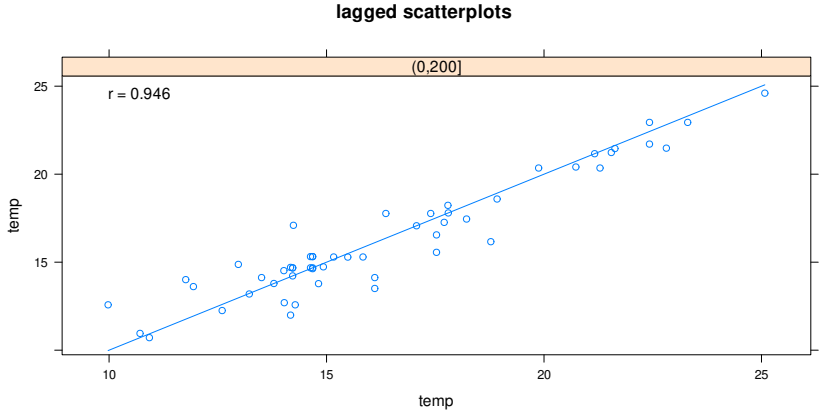
- $h$ - lag - odstęp pomiędzy dwoma lokalizacjami
- $z(s+h)$ - wartość badanej zmiennej odległej o odstęp $h$ - inaczej określany jako głowa (ang. *head*)

## 6.2. Przestrzenna kowariancja i korelacja

### 6.2.1. Miary podobieństwa

Przestrzenna kowariancja, korelacja i semiwariancja to miary określające przestrzenną zmienność analizowanej cechy.

- Kowariancja i korelacja to miary podobieństwa pomiędzy dwoma zmiennymi
- Przenosząc to na aspekt przestrzenny, badamy jedną zmienną, ale pomiędzy dwoma punktami odległymi od siebie o pewien dystans (określany jako lag)
- W efekcie otrzymujemy miarę podobieństwa pomiędzy wartością głowy i ogona

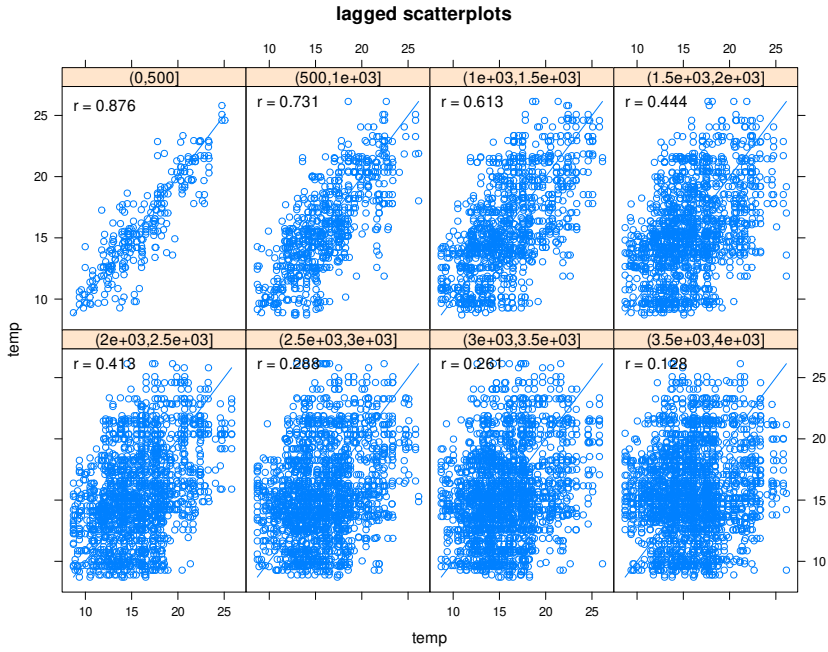



### 6.2.2. Wykres rozrzutu z przesunięciem

Wykres rozrzutu z przesunięciem pokazuje korelację pomiędzy wartościami analizowanej cechy w pewnych grupach odległości. Taki wykres można stworzyć używając funkcji `hscat()` z pakietu **gstat**. Przykładowo, na poniższym wykresie widać wartość cechy `temp` z kolejnymi przesunięciami - od 0 do 500 metrów, od 500 metrów do 1000 metrów, itd. W pierwszym przedziale wartość cechy `temp` z przesunięciem wykazuje korelację na poziomie 0,876, a następnie z każdym kolejnym przedziałem (odległością) ta

wartość maleje. W przedziale 3500 do 4000 metrów osiąga jedynie 0,128. Pozwala to na stwierdzenie, że cecha `temp` wykazuje zmienność przestrzenną - podobieństwo jej wartości zmniejsza się wraz z odległością.

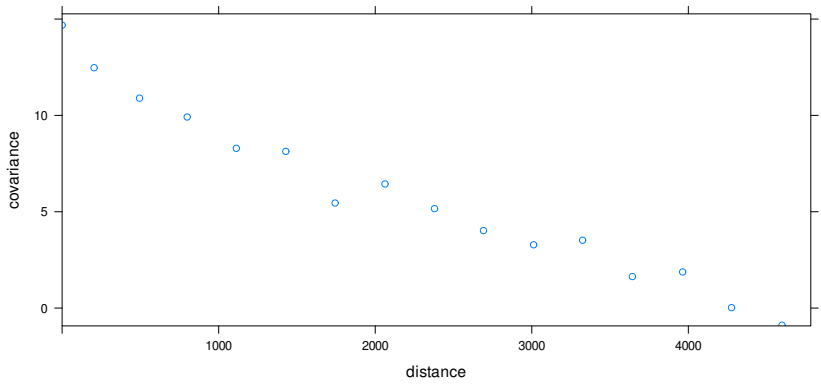
```
hscat(temp~1, data = punkty, breaks = seq(0, 4000, by = 500))
```




## 6.2.3. Autokowariancja

Podobną informację jak wykres rozrzutu z przesunięciem daje autokowariancja. Pokazuje ona jak mocno przestrzennie powiązane są wartości par obserwacji odległych od siebie o kolejne przedziały. Jej wyliczenie jest możliwe z użyciem funkcji `variogram()` z pakietu **gstat**, gdzie definiuje się analizowaną zmienną, zbiór punktowy, oraz ustala się argument `covariogram` na `TRUE`.

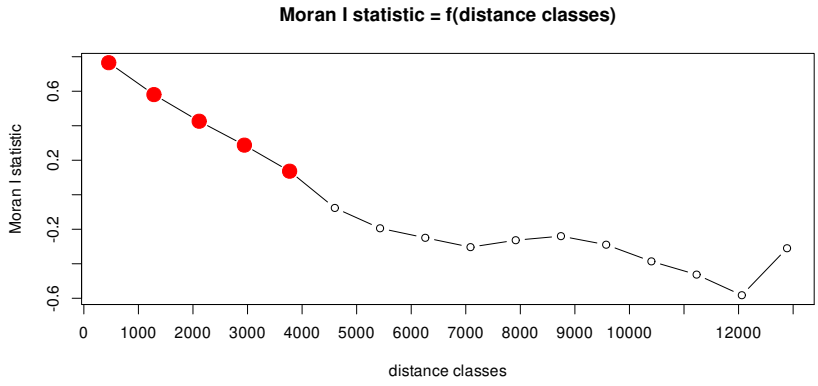
```
kowario <- variogram(temp~1, locations = punkty,
                     covariogram = TRUE)
plot(kowario)
```




### 6.2.4. Autokorelacja

Kolejną miarą przestrzennego podobieństwa jest autokorelacja. Jej wykres (autokorelogram) pokazuje wartość jednej z miar autokorelacji (np. I Morana lub C Geary'ego) w stosunku do odległości. Na poniższym przykładzie, wartość statystyki I Morana jest wyliczana poprzez funkcję `correlog()` z pakietu **pgirmess**.

```
wsp <- coordinates(punkty)
kor <- correlog(wsp, punkty$temp, method = "Moran")
plot(kor)
```



## 6.3. Semiwariancja

### 6.3.1. Miara niepodobieństwa

Trzecią miarę charakteryzującą relację między obserwacjami odległymi o kolejne odstępy jest semiwariancja. Z praktycznego punktu widzenia, semiwariogram jest preferowaną miarą relacji przestrzennej, ponieważ wykazuje tendencję do lepszego wygładzania danych niż funkcja kowariancji. Dodatkowo, semiwariogram jest mniej wymagający obliczeniowo. Warto jednak dodać, że dla potrzeb interpretacji relacji, kowarancja i korelacja przestrzenna nadaje się nie gorzej niż semiwariancja.

### 6.3.2. Wzór

Zmienność przestrzenna analizowanej cechy może być określona za pomocą semiwariancji. Jest to połowa średniej kwadratu różnicy pomiędzy wartościami badanej zmiennej w dwóch lokalizacjach odległych o wektor $h$:

$$\gamma(h) = \frac{1}{2}[(z(s) - z(s+h))]^2$$

Przykładowo, aby wyliczyć wartość semiwariancji (`gamma`) pomiędzy dwoma punktami musimy znać wartość pierwszego z nich (w przykładzie jest to ok. 22,46 stopni Celsjusza) oraz drugiego z nich (ok. 16,04 stopni Celsjusza). Korzystając z wzoru na semiwariację otrzymujemy wartość równą ok. 20,6. Znamy również odległość między punktami (ok. 4576,59 metra), więc możemy w uproszczeniu stwierdzić, że dla tej pary punktów odległych o 4577 metry wartość semiwariancji wynosi około 20,6.

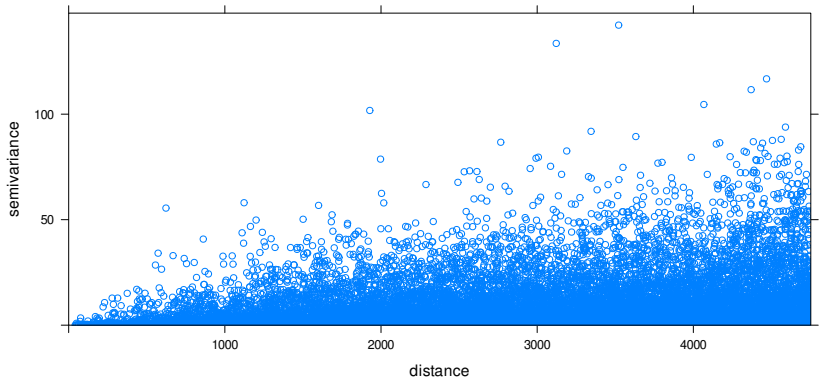
```
odl <- dist(coordinates(punkty)[c(1, 2), ])
gamma <- 0.5 * (punkty$temp[1] - punkty$temp[2]) ^ 2
gamma
```

```
## [1] 20.60122
```

### 6.3.3. Chmura semiwariogramu

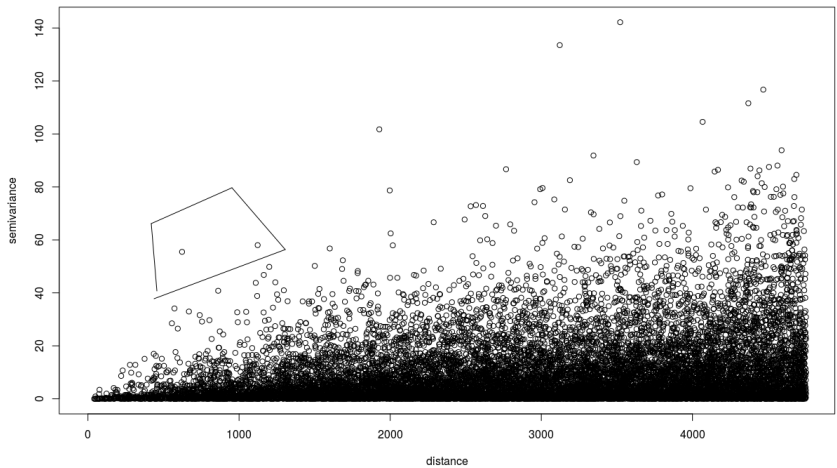
Jeżeli w badanej próbie mamy $n$ obserwacji oznacza to, że możemy zaobserwować $\frac{1}{2}n(n-1)$ par obserwacji. Każda z tych par obserwacji daje nam informację o semiwariancji występującej wraz z odległością. Tę semiwariancję można zaprezentować na wykresie zwanym chmurą semiwariogramu. Do jej wyliczenia służy funkcja `variogram()` z argumentem `cloud = TRUE`.

```
vario_cloud <- variogram(temp~1, locations = punkty,
                         cloud = TRUE)
plot(vario_cloud)
```



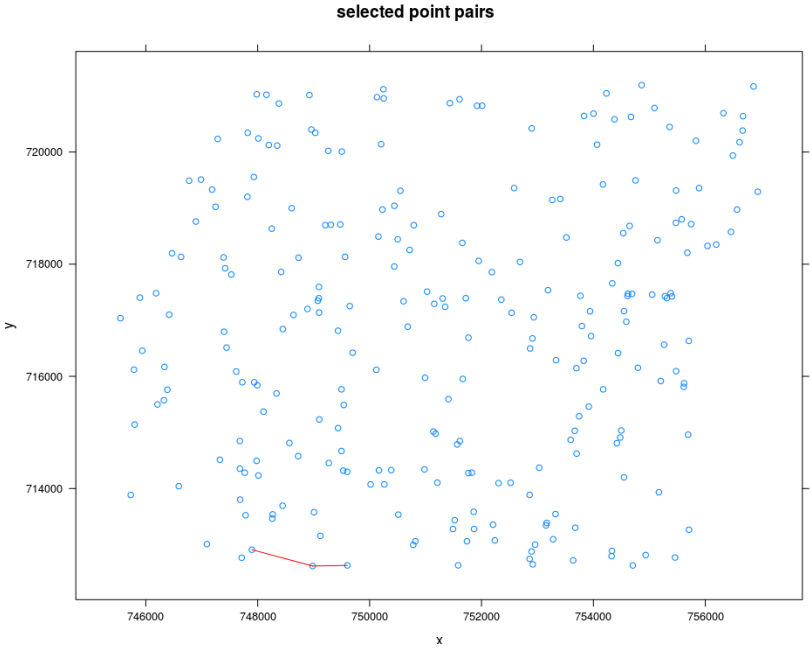

Chmura semiwariogramu pozwala także na zauważenie par punktów, których wartości znacząco odstają. Aby zlokalizować te pary punktów, można zastosować funkcję `plot()` z argumentem `digitize = TRUE`, a następnie za pomocą kilku kliknięć określić obszar zawierający nietypowe wartości. Po skończonym wyborze należy nacisnąć klawisz Esc.

```
vario_cloud_sel <- plot(vario_cloud, digitize = TRUE)
```



Następnie można wyświetlić wybrane pary punktów z użyciem funkcji `plot()`.

```
plot(vario_cloud_sel, punkty, "Spatial")
```



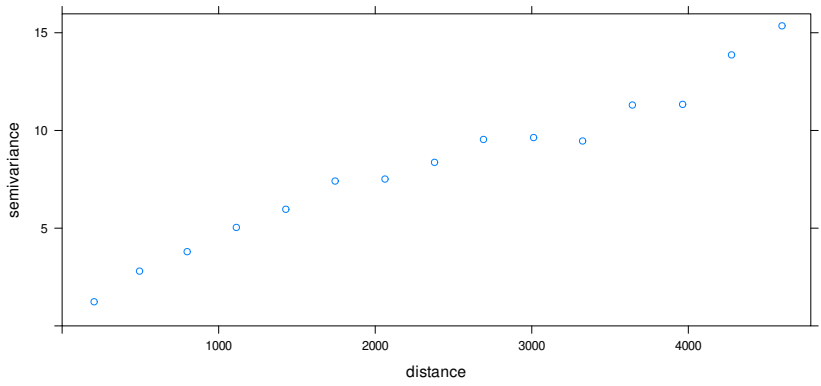

## 6.3.4. Charakterystyka struktury przestrzennej

Semiwariogram jest wykresem pokazującym relację pomiędzy odległością a semiwariancją. Inaczej mówiąc, dla kolejnych odstępów (lagów) wartość semiwariancji jest uśredniana i przestawiania w odniesieniu do odległości.

$$\hat{\gamma}(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} (z(s_i) - z(s_i + h))^2$$

, gdzie $N(h)$ oznacza liczbę par punktów w odstępie $h$.

W oparciu o semiwariogram empiryczny (czyli oparty na danych) możemy następnie dopasować do niego model/e.




### 6.3.5. Tworzenie semiwariogramu

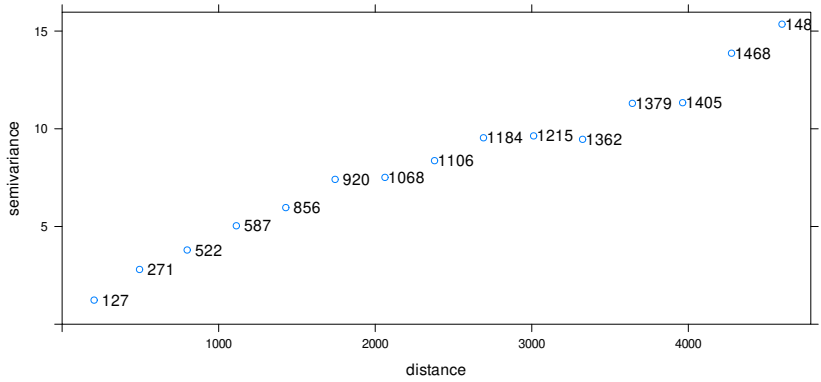
Stworzenie podstawowego semiwariogramu w pakiecie **gstat** odbywa się z użyciem funkcji `variogram()`. Należy w niej zdefiniować analizowaną zmienną (w tym przykładzie `temp~1`) oraz zbiór punktowy (`punkty`).

```
vario_par <- variogram(temp~1, locations = punkty)
vario_par
```

```
##       np      dist     gamma dir.hor dir.ver   id
## 1    127  204.2244  1.233999       0       0 var1
## 2    271  494.5572  2.802019       0       0 var1
## 3    522  798.7911  3.797612       0       0 var1
## 4    587 1112.7783  5.037545       0       0 var1
## 5    856 1428.7866  5.967276       0       0 var1
## 6    920 1743.7864  7.411276       0       0 var1
## 7   1068 2062.3041  7.514067       0       0 var1
## 8   1106 2378.9333  8.369087       0       0 var1
## 9   1184 2691.5206  9.539870       0       0 var1
## 10 1215 3011.7729  9.636891       0       0 var1
## 11 1362 3324.6705  9.461429       0       0 var1
## 12 1379 3642.5616 11.301515       0       0 var1
## 13 1405 3963.6776 11.335589       0       0 var1
## 14 1468 4276.0078 13.866888       0       0 var1
## 15 1482 4598.4144 15.353206       0       0 var1
```

Do wyświetlenia semiwariogramu służy funkcja `plot()`. Można również dodać informację o liczbie par punktów, jaka posłużyła do wyliczenia semiwariancji dla kolejnych odstępów poprzez argument `plot.numbers = TRUE`.

```
plot(vario_par, plot.numbers = TRUE)
```




### 6.3.6. Rules of thumb

Przy ustalaniu parametrów semiwariogramu powinno się stosować do kilku utartych zasad (tzw. *rules of thumb*):

- W każdym odstępie powinno się znaleźć co najmniej 30 par punktów
- Maksymalny zasięg semiwariogramu (ang. *cutoff distance*) to 1/2 pierwiastka z badanej powierzchni (inne źródła mówią o połowie z przekątnej badanego obszaru/jednej trzeciej)
- Liczba odstępów powinna nie być mniejsza niż 10
- Optymalnie maksymalny zasięg semiwariogramu powinien być dłuższy o 10-15% od zasięgu zjawiska
- Optymalnie odstępy powinny być jak najbliżej siebie i jednocześnie nie być chaotyczne
- Warto metodą prób i błędów określić optymalne parametry semiwariogramu
- Należy określić czy zjawisko wykazuje anizotropię przestrzenną

### 6.3.7. Obliczenia pomocnicze

- Liczba par obserwacji:

```
0.5 * nrow(punkty) * (nrow(punkty) - 1)
```

```
## [1] 31125
```

- Powierzchnia zajmowana przez jedną próbkę:

```
pow_pr <- gArea(granica) / nrow(punkty)
pow_pr
```

```
## [1] 253506.6
```

- Średnia odległość między punktami :

```
sqrt(pow_pr)
```

```
## [1] 503.4944
```

- Połowa pierwiastka powierzchni:

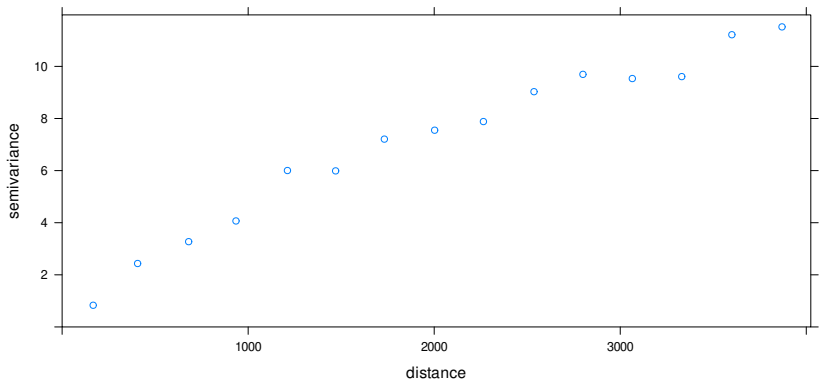
```
pow <- gArea(granica)
0.5 * sqrt(pow)
```

```
## [1] 3980.472
```

### 6.3.8. Modyfikacja semiwariogramu

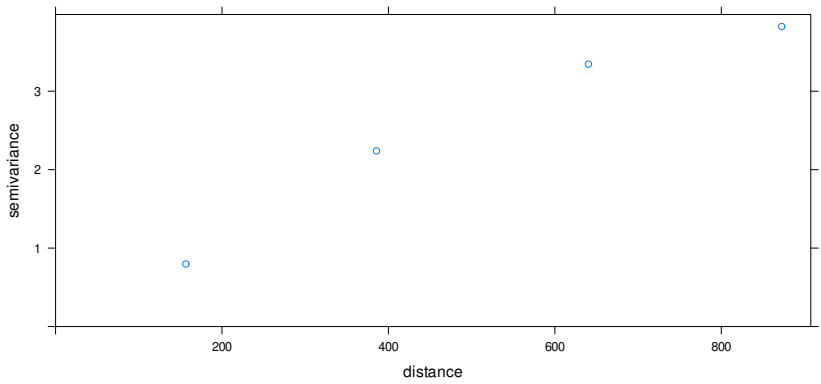
Maksymalny zasięg semiwariogramu (ang. *cutoff distance*) jest domyślnie wyliczany w pakiecie **gstat** jako 1/3 z najdłuższej przekątnej badanego obszaru. Można jednak tę wartość zmodyfikować używając argumentu `cutoff`.

```
vario_par <- variogram(temp~1, locations = punkty,
                       cutoff = 4000)
plot(vario_par)
```



Dodatkowo, domyślnie w pakiecie **gstat** odległość między przedziałami (ang. *interval width*) jest wyliczana jako maksymalny zasięg semiwariogramu podzielony przez 15. Można to oczywiście zmienić używając zarówno argumentu `cutoff`, jak i argumentu `width` mówiącego o szerokości odstępów.

```
vario_par <- variogram(temp~1, locations = punkty,
                       cutoff = 1000, width = 250)
plot(vario_par)
```




## 6.4. Anizotropia

### 6.4.1. Anizotropia struktury przestrzennej

W wielu sytuacjach, wartość cechy zależy nie tylko od odległości, ale także od kierunku. O takim zjawisku mówimy, że wykazuje ono anizotropię struktury przestrzennej.

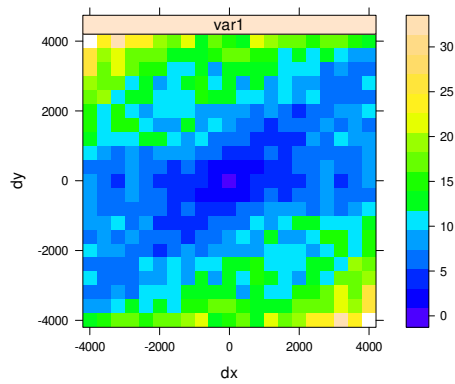
### 6.4.2. Mapa semiwariogramu

Mapa semiwariogramu (zwana inaczej powierzchnią semiwariogramu) służy do określenia czy struktura przestrzenna zjawiska posiada anizotropię, a jeżeli tak to w jakim kierunku. Na podstawie mapy semiwariogramu identyfikuje się także parametry potrzebne do zbudowania semiwariogramów kierunkowych.

Stworzenie mapy semiwariogramu odbywa się poprzez dodanie kilku argumentów do funkcji `variogram()`: oprócz argumentu zmiennej i zbioru punktowego, jest to `cutoff`, `width`, oraz `map = TRUE`. Następnie można ją wyświetlić używając funkcji `plot()`. Warto w tym wypadku użyć dodatkowego

argumentu `threshold`, który powoduje, że niepewna wartość semiwariancji (wyliczona na małej liczbie punktów) nie jest wyświetlana.

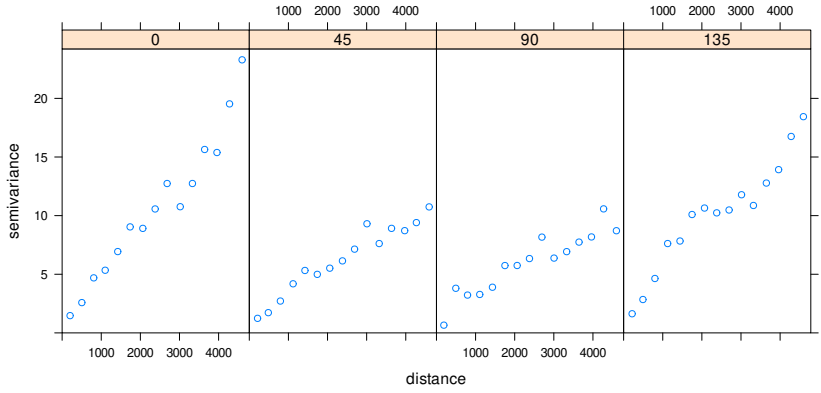
```
vario_map <- variogram(temp~1, locations = punkty,
                       cutoff = 4000, width = 400, map = TRUE)
plot(vario_map, threshold = 30,
     col.regions = topo.colors(n = 40)) # co najmniej 30 par punktów
```




### 6.4.3. Semiwariogramy kierunkowe

W przypadku, gdy zjawisko wykazuje anizotropię przestrzenną, możliwe jest stworzenie semiwariogramów dla różnych wybranych kierunków. Dla argumentu `alpha = c(0, 45, 90, 135)` wybrane cztery główne kierunki to 0, 45, 90 i 135 stopni. Oznacza to, że przykładowo dla kierunku 45 stopni brane pod uwagę będą wszystkie pary punktów pomiędzy 22,5 a 67,5 stopnia.

```
vario_kier <- variogram(temp~1, locations = punkty,
                        alpha = c(0, 45, 90, 135))
plot(vario_kier)
```




## 6.5. Zadania

Przyjrzyj się danym z obiektu `punkty_pref`. Możesz go wczytać używając poniższego kodu:

```
data(punkty_pref)
```

1. Stwórz wykres rozrzutu z przesunięciem dla zmiennej `srtm` dla odstępów od 0 do 5000 metrów co 625 metry. Co można odczytać z otrzymanego wykresu?
2. Wylicz odległość oraz wartość semiwariancji dla zmiennej `srtm` pomiędzy pierwszym i drugim, pierwszym i trzecim, oraz drugim i trzecim punktem. Zwizualizuj te trzy punkty. W jaki sposób można zinterpretować otrzymane wyniki wartości semiwariancji?
3. Twórz chmurę semiwariogramu dla zmiennej `temp`. W jaki sposób można zaobserwować na niej wartości odstające? Co one oznaczają?
4. Jaka jest liczba par obserwacji, średnia powierzchnia zajmowana przez jedną próbkę (w $km^2$) oraz średnia odległość między punktami (w km)?
5. Stwórz semiwariogram zmiennej `srtm`. Ile par punktów posłużyło do wyliczenia pierwszego odstępu?
6. Zmodyfikuj powyższy semiwariogram, żeby jego maksymalny zasięg wynosił 3,5 kilometra.
7. Stwórz mapę semiwariogramu dla zmiennej `srtm`. Sprawdź różne wartości `cutoff` (od 3 do 5 km) oraz `width` (od 200 do 500 metrów). Zmień domyślną paletę kolorów w wizualizacji mapy semiwariogramu. (Dodatkowe: spróbuj do tego celu użyć pakietu **viridisLite**).

8. Co przedstawia stworzona mapa semiwariogramu? Czy badane zjawisko wykazuje izotropia czy anizotropia?
9. Jeżeli mapa semiwariogramu wykazuje anizotropię struktury przestrzennej to w jakim kierunku? Stwórz semiwariogramy dla wybranych kierunków.

# 7. Modelowanie autokorelacji przestrzennej

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(geostatbook)
data(punkty)
```

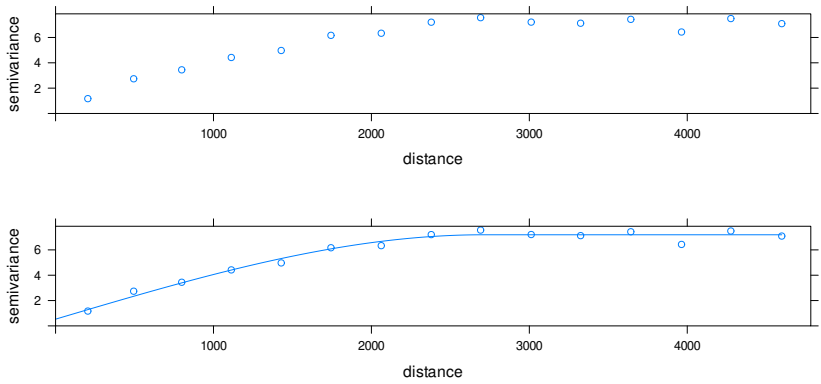
## 7.1. Modelowanie matematycznie autokorelacji przestrzennej

### 7.1.1. Modelowanie struktury przestrzennej

Semiwariogram empiryczny (wyliczony z danych punktowych) jest:

- Nieciągły - wartości semiwariancji są średnimi przedziałowymi
- Chaotyczny - badana próba jest jedynie przybliżeniem rzeczywistości, dodatkowo obciążonym błędami

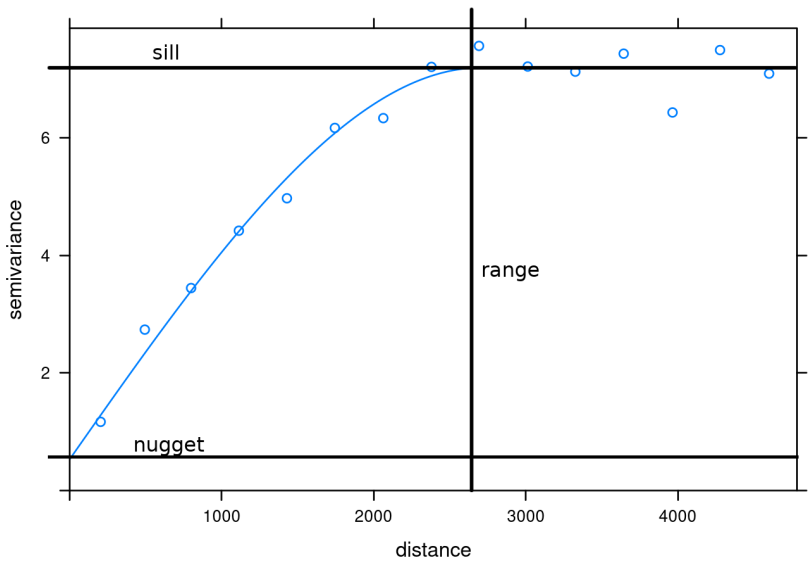
Estymacje i symulacje przestrzenne wymagają modelu struktury przestrzennej analizowanej cechy, a nie tylko wartości empirycznych. Dodatkowo, matematycznie modelowanie wygładza chaotyczne fluktuacje danych empirycznych.




### 7.1.2. Model semiwariogramu

Model semiwariogramu składa się zazwyczaj z trzech podstawowych elementów. Są to:

- **Nugget** - efekt nuggetowy - pozwala na określenie błędu w danych wejściowych oraz zmienności na dystansie krótszym niż pierwszy odstęp
- **Sill** - semiwariancja progowa - oznacza wariancję badanej zmiennej
- **Range** - zasięg - to odległość do której istnieje przestrzenna korelacja



### 7.1.3. Model nuggetowy

Model nuggetowy określa sytuację, w której analizowana zmienna nie wykazuje autokorelacji. Inaczej mówiąc, niepodobieństwo jej wartości nie wzrasta wraz z odległością. Model nuggetowy nie powinien być używany samodzielnie - w większości zastosowań jest on elementem modelu złożonego. Służy on do określania, między innymi, błędu pomiarowego czy zmienności na krótkich odstępach.

## 7.2. Modele podstawowe

### 7.2.1. Typy modeli podstawowych

Pakiet `gstat` zawiera 20 podstawowych modeli geostatystycznych, w tym najczęściej używane takie jak:

- Nuggetowy (ang. *Nugget effect model*)
- Sferyczny (ang. *Spherical model*)
- Gaussowski (ang. *Gaussian model*)
- Potęgowy (ang. *Power model*)
- Wykładniczy (ang. *Exponential model*)
- Inne

Do wyświetlenia listy nazw modeli i ich skrótów służy funkcja `vgm()`.

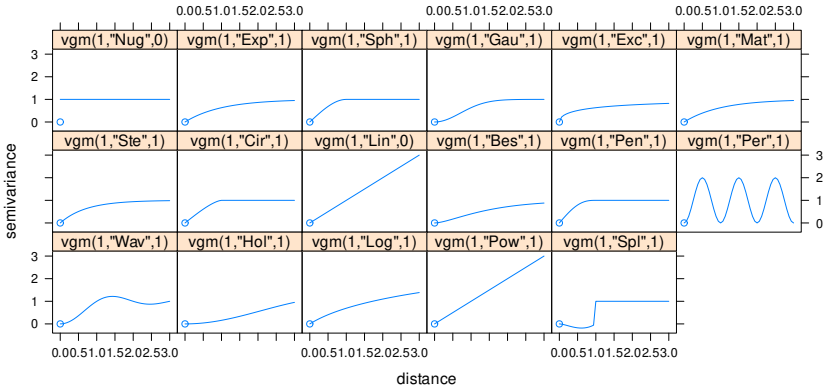
```
vgm()
```

```
##    short                                     long
## 1    Nug                              Nug (nugget)
## 2    Exp                         Exp (exponential)
## 3    Sph                           Sph (spherical)
## 4    Gau                            Gau (gaussian)
## 5    Exc        Exclass (Exponential class/stable)
## 6    Mat                              Mat (Matern)
## 7    Ste Mat (Matern, M. Stein's parameterization)
## 8    Cir                            Cir (circular)
## 9    Lin                              Lin (linear)
## 10   Bes                              Bes (bessel)
## 11   Pen                     Pen (pentaspherical)
## 12   Per                            Per (periodic)
## 13   Wav                                Wav (wave)
## 14   Hol                                Hol (hole)
## 15   Log                       Log (logarithmic)
## 16   Pow                             Pow (power)
```

```
## 17   Spl                              Spl (spline)
## 18   Leg                            Leg (Legendre)
## 19   Err                 Err (Measurement error)
## 20   Int                         Int (Intercept)
```

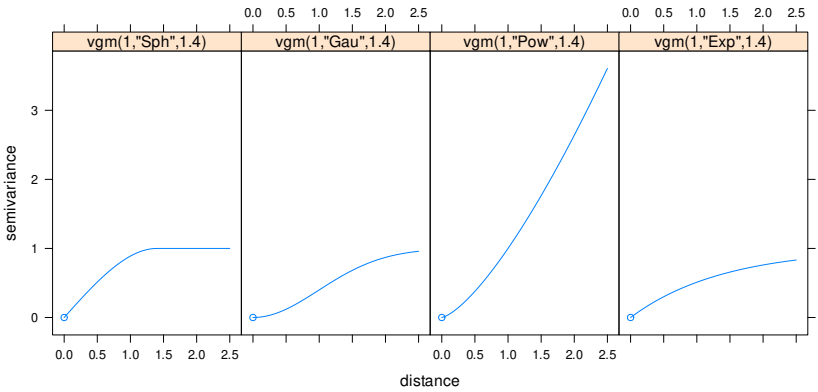
Można się również im przyjrzeć używając funkcji `show.vgms()`.

```
show.vgms()
```



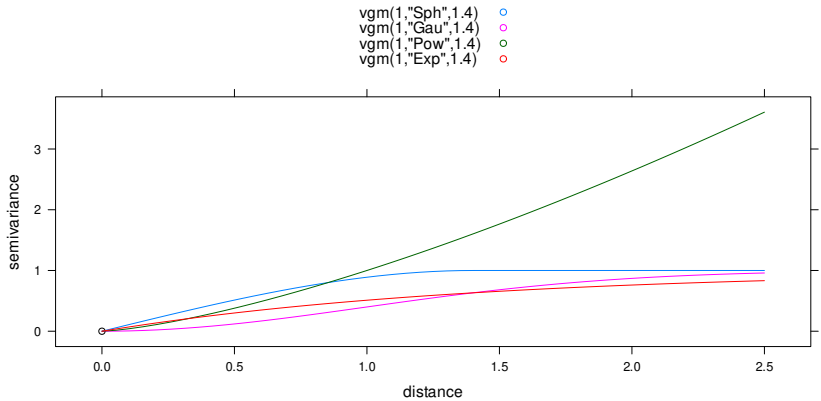

Istnieje możliwość wyświetlenia tylko wybranych modeli podstawowych poprzez argument `models`.

```
show.vgms(models = c("Sph", "Gau", "Pow", "Exp"),
          range = 1.4, max = 2.5)
```



Dodatkowo, można je porównać na jednym wykresie poprzez argument `as.groups = TRUE`.

```
show.vgms(models = c("Sph", "Gau", "Pow", "Exp"),
          range = 1.4, max = 2.5, as.groups = TRUE)
```




## 7.3. Metody modelowania

### 7.3.1. Rodzaje metod modelowania

Istnieją trzy najczęściej spotykane metody modelowania geostatystycznego:

- Ustawianie "ręczne" parametrów modelu, np. funkcja `vgm()` z pakietu **gstat**
- Ustawianie "wizualne" parametrów modelu, np. funkcja `eyefit()` z pakietu **geoR**
- Automatyczny wybór parametrów na podstawie różnych kryteriów statystycznych, np. funkcja `fit.variogram()` z pakietu **gstat()**, `variofit()` z pakietu **geoR**, `autofitVariogram()` z pakietu **automap**

Odpowiednie określenie modelu matematycznego często nie jest proste. W efekcie automatyczne metody nie zawsze są w stanie dać lepszy wynik od modelowania "ręcznego". Najlepiej, gdy wybór modelu oparty jest o wiedzę na temat zakładanego procesu przestrzennego.

### 7.3.2. Funkcja `fit.variogram()`

Funkcja `fit.variogram()` z pakietu **gstat** dopasowuje zasięg oraz semiwariancję progową w oparciu o ustalone "ręcznie" wejściowe parametry modelu. Więcej na temat działania tej funkcji można przeczytać we wpisie Fitting variogram models in gstat[1] na stronie https://www.r-spatial.org.

## 7.4. Modelowanie izotropowe

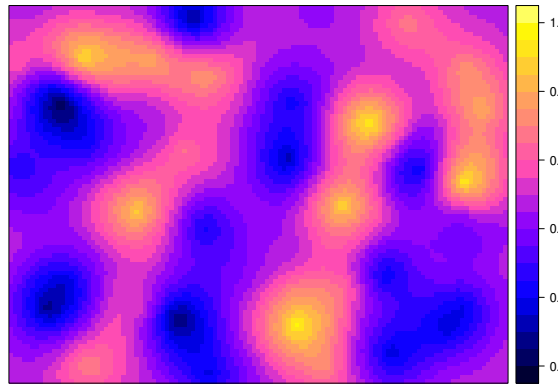
Do zbudowania modelu semiwariogramu należy wykonać szereg kroków:

1. Stworzyć i wyświetlić semiwariogram empiryczny analizowanej zmiennej z użyciem funkcji `variogram()` oraz `plot()`.
2. Zdefiniować wejściowe parametry semiwariogramu. W najprostszej sytuacji wystarczy zdefiniować używany model/e poprzez skróconą nazwę używanej funkcji (`model`). Możliwe, ale nie wymagane jest także określenie wejściowej semiwariancji cząstkowej (`psill`) oraz zasięgu modelu (`range`) w funkcji `vgm()`. Uzyskany model można przedstawić w funkcji `plot()` podając nazwę obiektu zawierającego semiwariogram empiryczny oraz obiektu zawierającego model.
3. Dopasować parametry modelu używając funkcji `fit.variogram()`. To dopasowanie można również zwizualizować używając funkcji `plot()`.

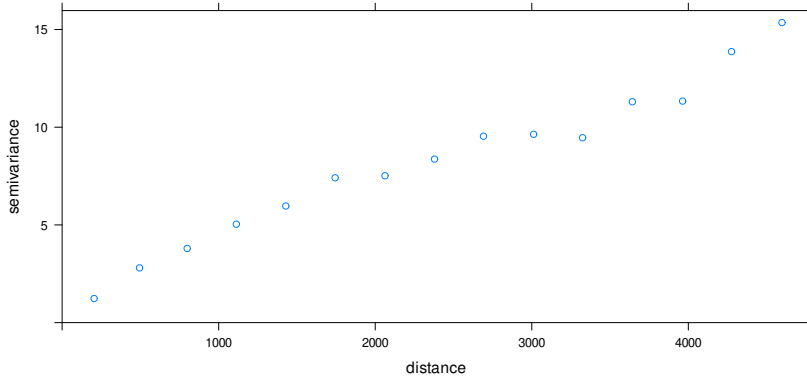
### 7.4.1. Model sferyczny

Model sferyczny (`Sph`) jest jednym z najczęściej stosowanych modeli geostatystycznych. Reprezentuje on cechę, której zmienność wartości ma charakter naprzemiennych płatów niskich i wysokich wartości. Średnio te płaty mają średnicę określoną przez zasięg (`range`) modelu.

[1] https://www.r-spatial.org/r/2016/02/14/gstat-variogram-fitting.html



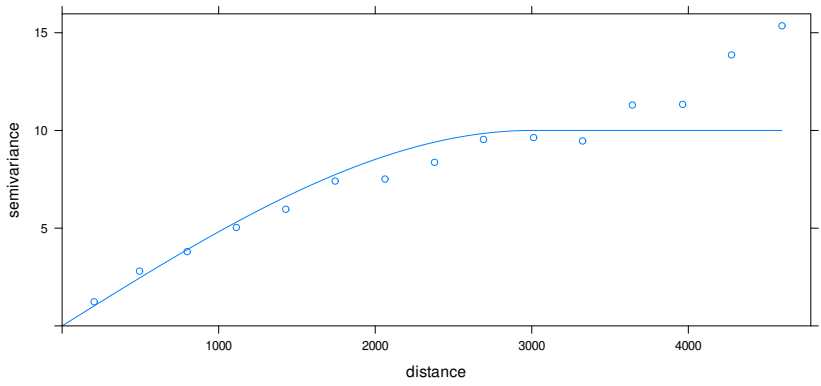

```
vario <- variogram(temp~1, locations = punkty)
plot(vario)
```




```
# ręczne ustalenie wszystkich parametrów modelu
model_sph <- vgm(psill = 10, model = "Sph", range = 3000)
model_sph
```

```
##   model psill range
## 1   Sph    10  3000
```

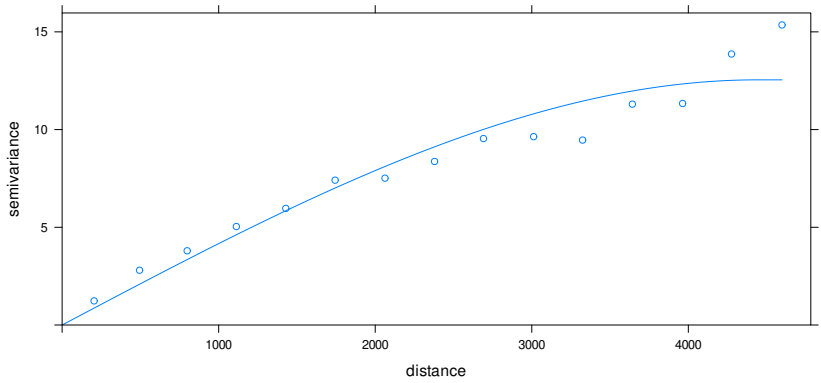
```
plot(vario, model = model_sph)
```



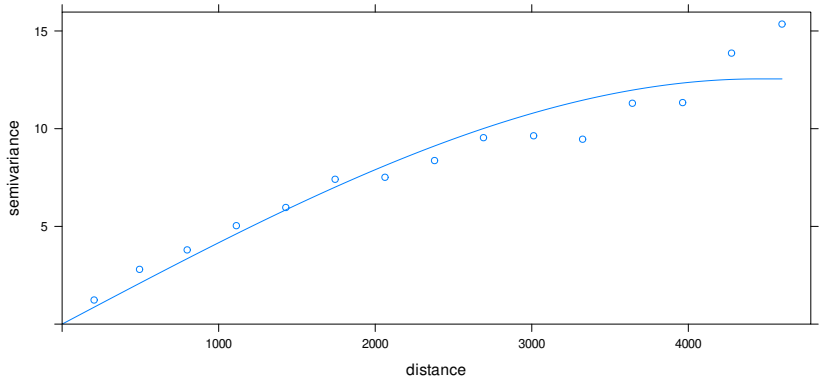

```
fitted_sph <- fit.variogram(vario, model_sph)
fitted_sph
```

```
##   model    psill    range
## 1   Sph 12.5445 4440.768
```

```
plot(vario, model = fitted_sph)
```

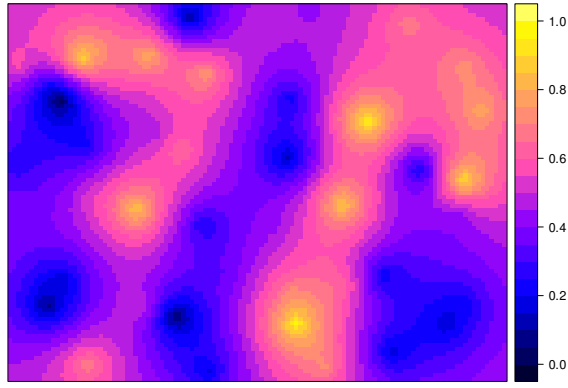



```
# ręczne ustalenie tylko typu modelu
model_sph2 <- vgm(model = "Sph")
fitted_sph2 <- fit.variogram(vario, model_sph2)
plot(vario, model = fitted_sph2)
```

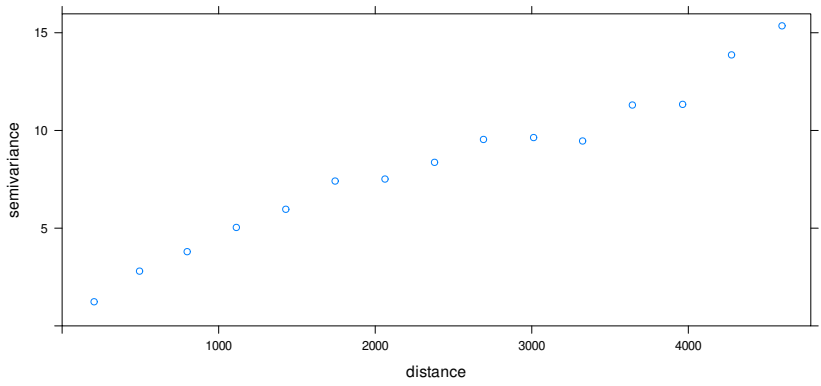



## 7.4.2. Model wykładniczy

Model wykładniczy (`Exp`) również jest jednym z najczęściej używanych w geostatystyce. Od modelu sferycznego różni go szczególnie to, że nie ma on skończonego zasięgu. W jego przypadku, zamiast zasięgu podaje się tzw. zasięg praktyczny. Oznacza on odległość na jakiej model osiąga 95% wartości wariancji progowej.



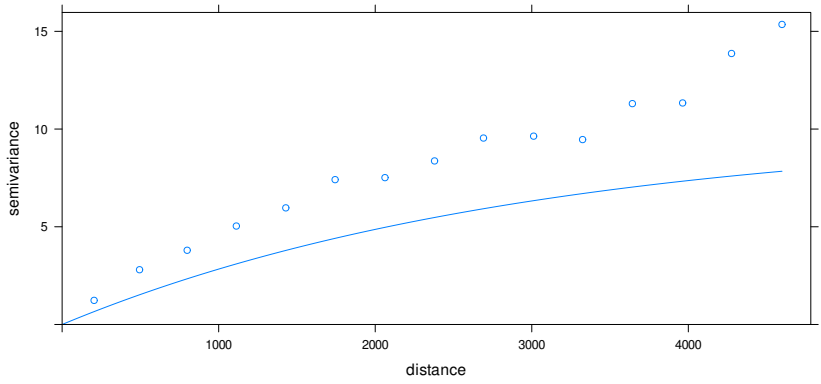

```
vario <- variogram(temp~1, locations = punkty)
plot(vario)
```




```
model_exp <- vgm(psill = 10, model = "Exp", range = 3000)
model_exp
```

```
##   model psill range
## 1   Exp    10  3000
```

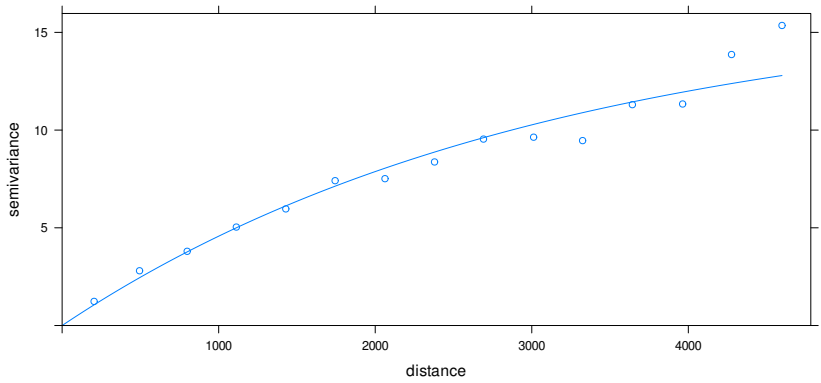
```
plot(vario, model = model_exp)
```




```
fitted_exp <- fit.variogram(vario, model_exp)
fitted_exp
```

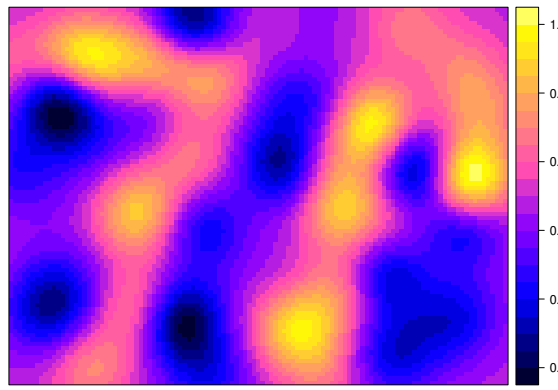
```
##   model    psill    range
## 1   Exp 16.50871 3084.309
```

```
plot(vario, model = fitted_exp)
```

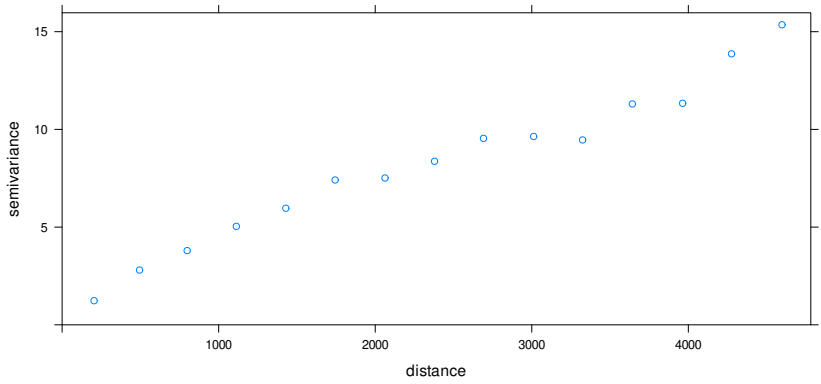



### 7.4.3. Model gaussowski

Model gaussowski (`Gau`) również posiada zasięg praktyczny definiowany jako 95% wartości wariancji progowej. Jego cechą charakterystyczną jest paraboliczny kształt na początkowym odcinku. Jest on najczęściej używany do modelowania cech o regularnej i łagodnej zmienności przestrzennej. Model gaussowski z uwagi na swoje cechy zazwyczaj nie powinien być stosowany samodzielnie, lecz jako element modelu złożonego.



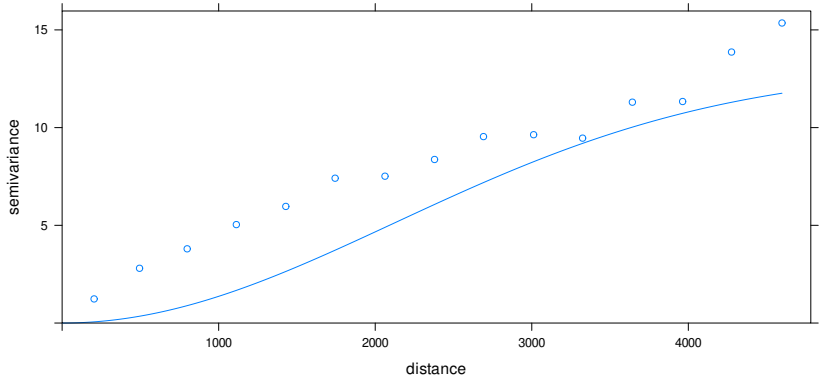
```
vario <- variogram(temp~1, locations = punkty)
plot(vario)
```




```
model_gau <- vgm(psill = 13, model = "Gau", range = 3000)
model_gau
```

```
##   model psill range
## 1   Gau    13  3000
```

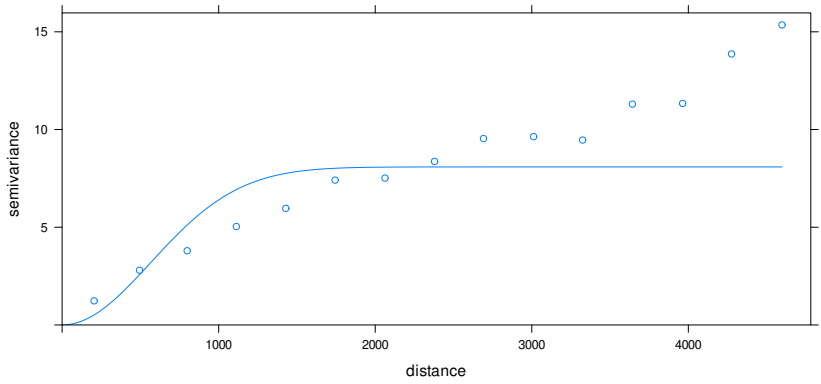
```
plot(vario, model = model_gau)
```



```
fitted_gau <- fit.variogram(vario, model_gau)
fitted_gau
```

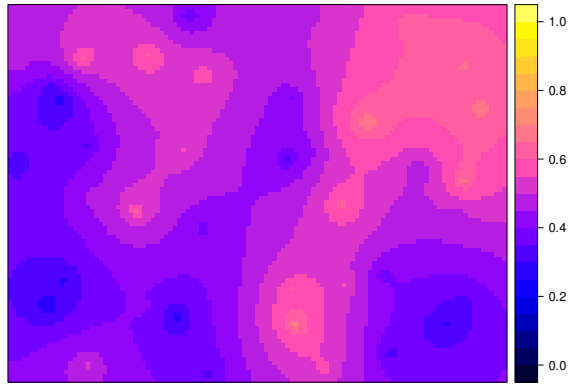
```
##   model    psill    range
## 1   Gau 8.085963 798.7454
```

```
plot(vario, model = fitted_gau)
```

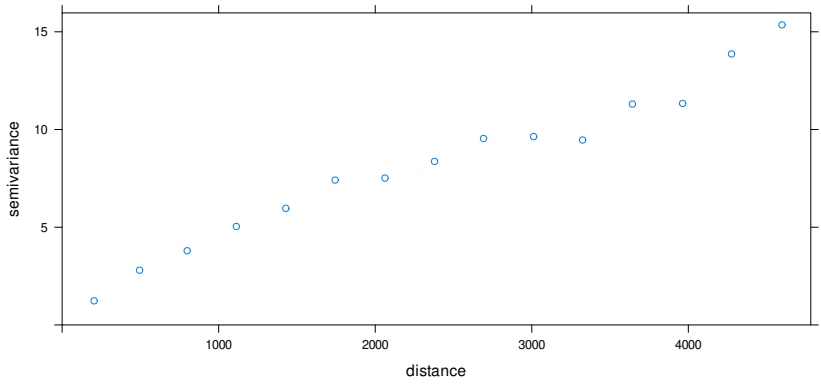



## 7.4.4. Model potęgowy

Model potęgowy (`Pow`) to przykład tzw. modelu nieograniczonego. Jego wartość rośnie w nieskończoność, dlatego niemożliwe jest określenie jego zasięgu. W przypadku modelu potęgowego, parametr `range` oznacza wykładnik potęgowy.



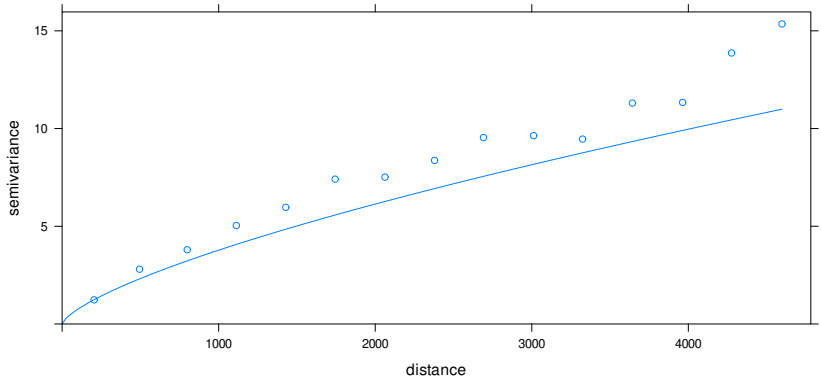
```
vario <- variogram(temp~1, locations = punkty)
plot(vario)
```




```
model_pow <- vgm(psill = 0.03, model = "Pow", range = 0.7)
model_pow
```

```
##   model psill range
## 1   Pow  0.03   0.7
```

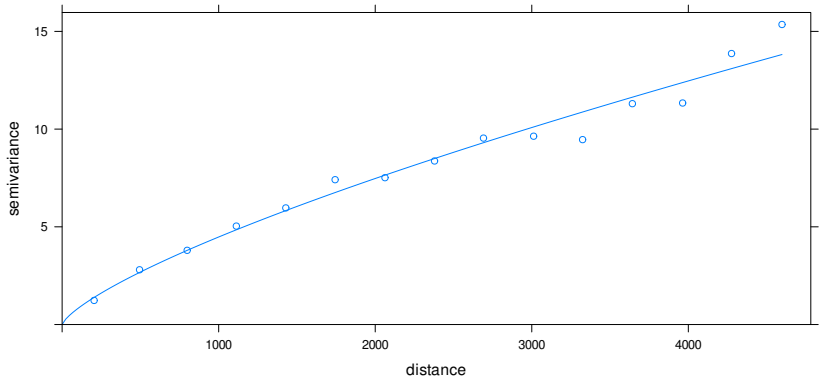
```
plot(vario, model = model_pow)
```



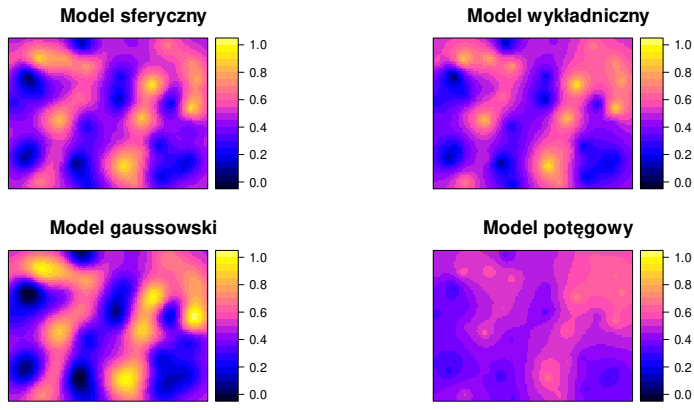
```
fitted_pow <- fit.variogram(vario, model_pow)
fitted_pow
```

```
##   model      psill     range
## 1   Pow 0.02732273 0.7382382
```

```
plot(vario, model = fitted_pow)
```




## 7.4.5. Porównanie modeli



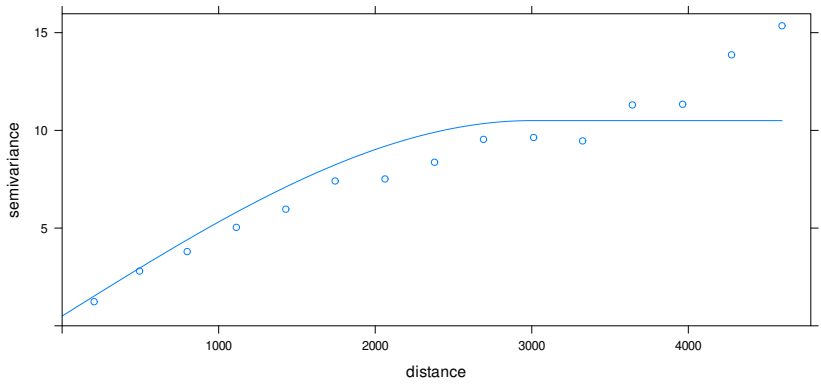
### 7.4.6. Modele złożone I

Najczęściej pojedynczy model nie jest w stanie odwzorować dokładnie zmienności przestrzennej analizowanej cechy. W takich sytuacjach konieczne jest połączenie dwóch lub więcej modeli podstawowych. Najbardziej powszechny model złożony składa się z funkcji nuggetowej (dla odległości zero) oraz drugiej funkcji (dla dalszej odległości). Zdefiniowanie takiej funkcji odbywa się poprzez dodanie argumentu `nugget` w funkcji `vgm()`.

```
vario <- variogram(temp~1, locations = punkty)
model_zl1 <- vgm(psill = 10, model = "Sph", range = 3000,
                 nugget = 0.5)
model_zl1
```

```
##   model psill range
## 1   Nug   0.5     0
## 2   Sph  10.0  3000
```

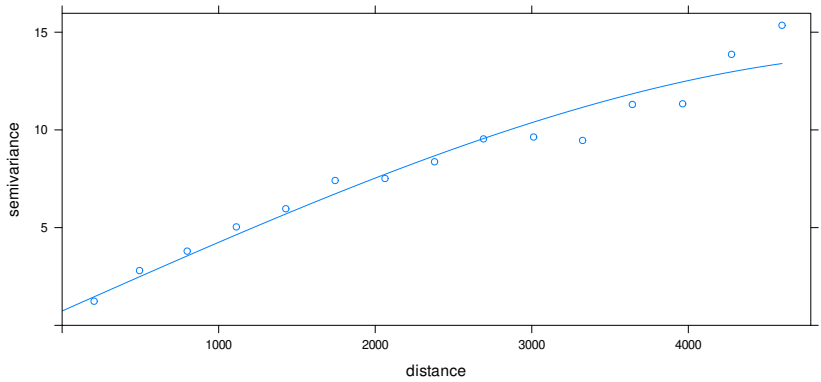
```
plot(vario, model = model_zl1)
```




```
fitted_zl1 <- fit.variogram(vario, model_zl1)
fitted_zl1
```

```
##   model      psill    range
## 1   Nug  0.7346354    0.000
## 2   Sph 13.2621876 5602.027
```

```
plot(vario, model = fitted_zl1)
```



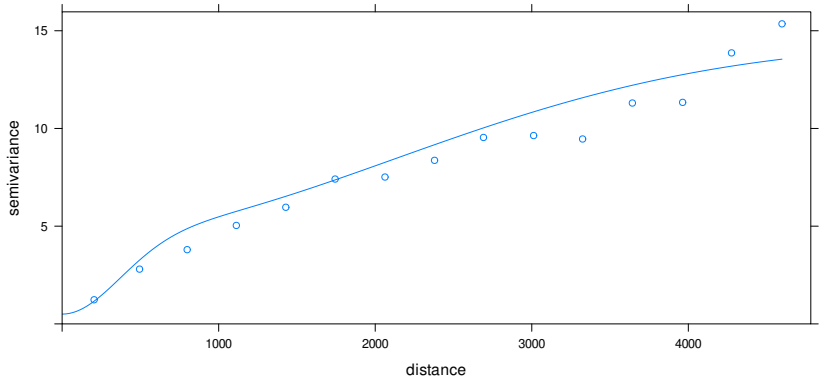

### 7.4.7. Modele złożone II

Bardziej złożone modele można tworzyć z pomocą argumentu `add.to`. Przyjmuje on kolejny obiekt funkcji `vgm()` i poprzez połączenie tych dwóch obiektów otrzymuje model złożony. Na poniższym przykładzie stworzony został model złożony składający się z modelu nuggetowego oraz dwóch modeli gaussowskich.

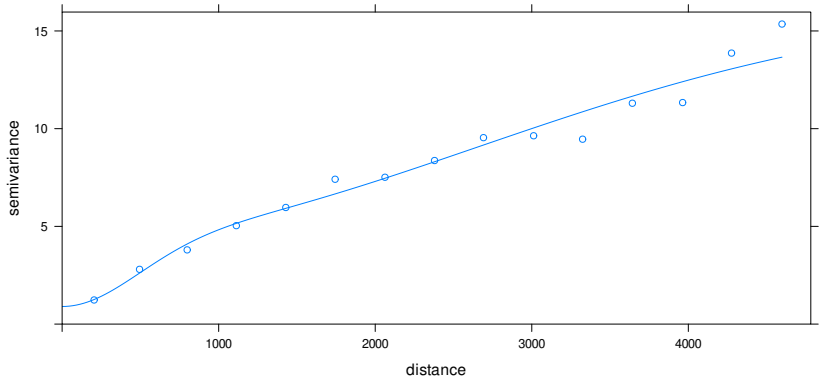
```
vario <- variogram(temp~1, locations = punkty)
model_zl2 <- vgm(10, "Gau", 3000, add.to = vgm(4,
                                                model = "Gau",
                                                range = 500,
                                                nugget = 0.5))
model_zl2
```

```
##   model psill range
## 1   Nug   0.5     0
## 2   Gau   4.0   500
## 3   Gau  10.0  3000
```

```
plot(vario, model = model_zl2)
```



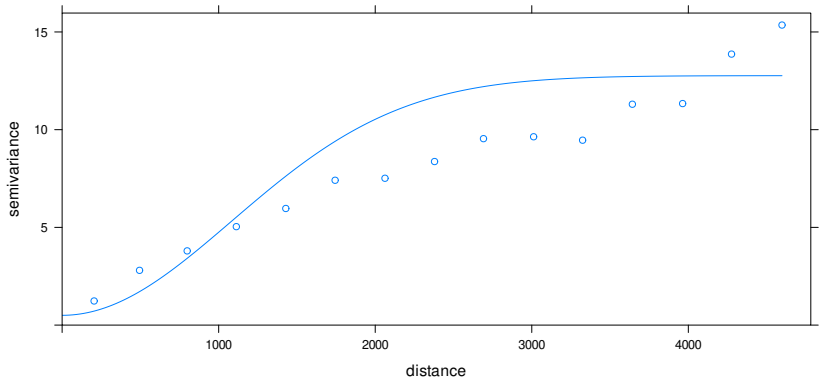

```
fitted_zl2 <- fit.variogram(vario, model_zl2)
plot(vario, model = fitted_zl2)
```




```
model_zl3 <- vgm(model = "Gau", add.to = vgm(model = "Gau",
                                             nugget = 0.5))
fitted_zl3 <- fit.variogram(vario, model_zl3)
fitted_zl3
```

```
##   model    psill    range
## 1   Nug 0.500000    0.000
## 2   Gau 6.131863 1532.805
## 3   Gau 6.131863 1532.805
```

```
plot(vario, model = fitted_zl3)
```

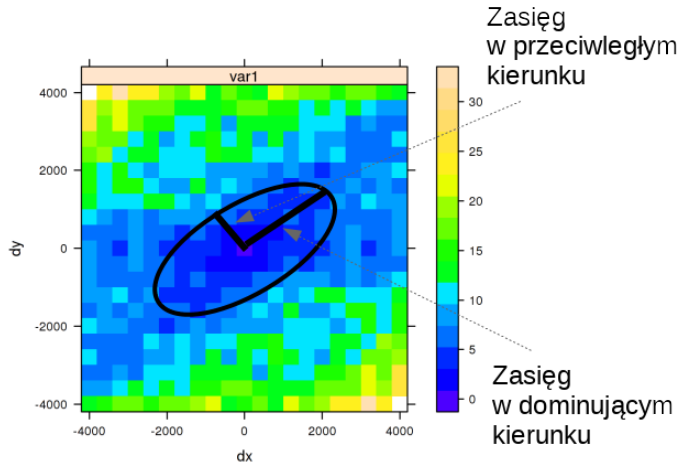



## 7.5. Modelowanie anizotropowe

### 7.5.1. Anizotropia

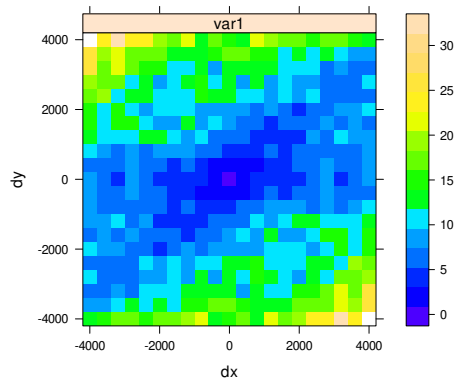
Uwzględnienie anizotropii wymaga zamiany parametru zasięgu na trzy inne parametry:

- Kąt określający dominujący kierunek
- Zasięg w dominującym kierunku
- Proporcję anizotropii, czyli relację pomiędzy zasięgiem w przeciwległym kierunku do zasięgu w dominującym kierunku

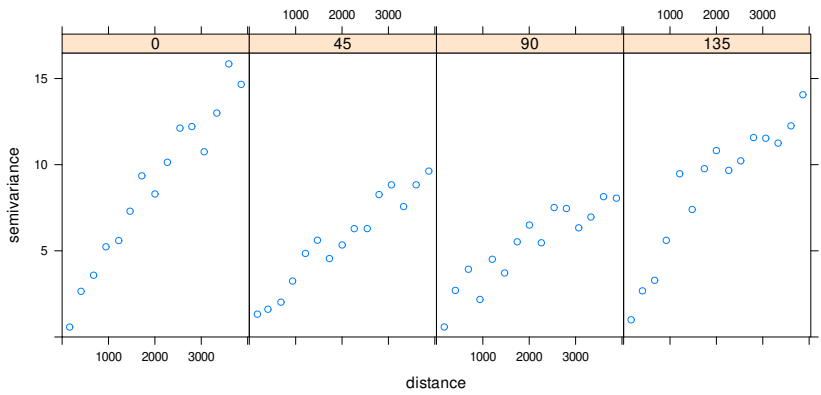



W pakiecie **gstat** odbywa się to poprzez dodanie argumentu `alpha` do funkcji `variogram()`. Należy w niej zdefiniować analizowane kierunki, które zostały określone na podstawie mapy semiwariogramu. Następnie w funkcji `vgm()` należy podać nowy argument `anis`. Przyjmuje on dwie wartości. Pierwsza z nich (`45` w przykładzie poniżej) oznacza dominujący kierunek anizotropii, druga zaś (`0.4`) mówi o tzw. proporcji anizotropii. Proporcja anizotropii jest to relacja pomiędzy zmiennością na kierunku prostopadłym a głównym kierunku. Na poniższym przykładzie zasięg ustalony dla głównego kierunku wynosi 4000 metrów. Wartość proporcji anizotropii, `0.4`, w tym wypadku oznacza że dla prostopadłego kierunku zasięg będzie wynosił 1600 metrów (4000 metrów x 0.4).

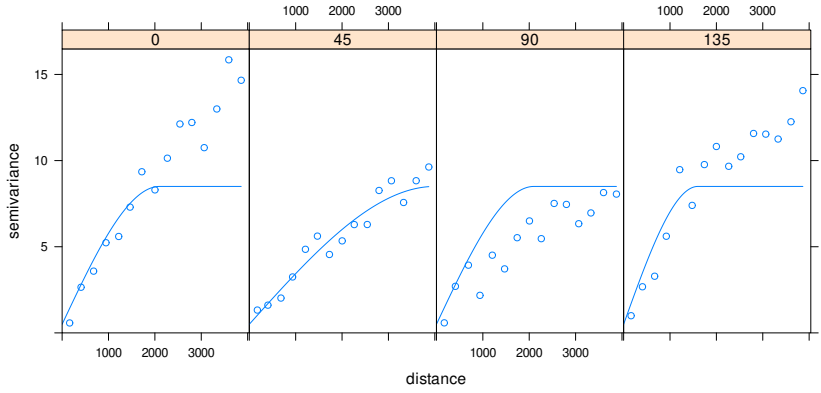
```
vario_map <- variogram(temp~1,
                       locations = punkty,
                       cutoff = 4000,
                       width = 400,
                       map = TRUE)
plot(vario_map, threshold = 30, col.regions = topo.colors(n = 40))
```



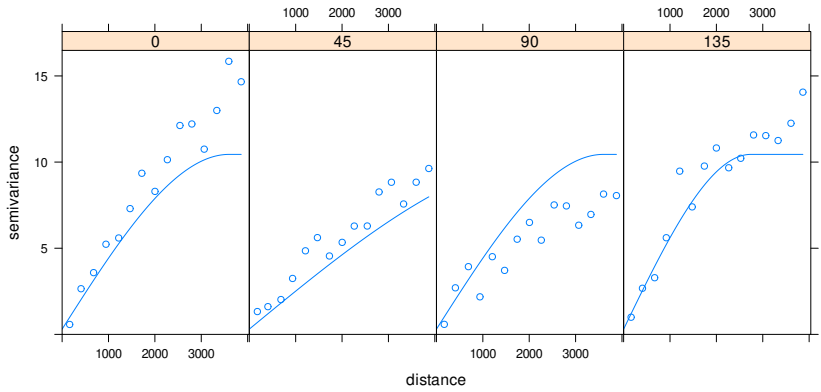

```
vario_kier <- variogram(temp~1,
                        locations = punkty,
                        alpha = c(0, 45, 90, 135),
                        cutoff = 4000)
plot(vario_kier)
```




```
vario_kier_fit <- vgm(psill = 8, model = "Sph", range = 4000,
                      nugget = 0.5, anis = c(45, 0.4))
plot(vario_kier, vario_kier_fit, as.table = TRUE)
```




```
vario_kier_fit2 <- fit.variogram(vario_kier,
                                 vgm(model = "Sph",
                                     anis = c(45, 0.4),
                                     nugget = 0.5))
plot(vario_kier, vario_kier_fit2, as.table = TRUE)
```




## 7.6. Zadania

Przyjrzyj się danym z obiektu `punkty_pref`. Możesz go wczytać używając poniższego kodu:

```
data(punkty_pref)
```

1. Zbuduj modele semiwariogramu zmiennej `srtm` używając modelu sferycznego używając zarówno ręcznie ustalonych parametrów oraz funkcji `fit.variogram()`. Porównaj graficznie uzyskane modele.
2. Zbuduj modele semiwariogramu zmiennej `srtm` używając modelu nuggetowego, sferycznego, wykładniczego, gausowskiego i potęgowego. Porównaj graficznie uzyskane modele.
3. Stwórz złożony model semiwariogramu zmiennej `srtm` używając modelu nuggetowego i sferycznego.
4. W oparciu o mapę semiwariogramu, zbuduj semiwariogramy kierunkowe zmiennej `srtm` dla kierunków wykazujących anizotropię przestrzenną. Następnie zbuduj modele semiwariogramu dla uzyskanych semiwariogramów kierunkowych.
5. (Dodatkowe) Spróbuj użyć jednego z modeli podstawowych, który nie był opisywany w tym rozdziale. Czym ten wybrany model się charakteryzuje?

# 8. Estymacje jednozmienne

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(geostatbook)
data(punkty)
data(siatka)
```

## 8.1. Kriging

### 8.1.1. Interpolacja geostatystyczna

Kriging (interpolacja geostatystyczna) to grupa metod estymacji zaproponowana w latach 50. przez Daniela Krige. Główna zasada mówi, że prognoza w danej lokalizacji jest kombinacją obokległych obserwacji. Waga nadawana każdej z obserwacji jest zależna od stopnia (przestrzennej) korelacji - stąd też bierze się istotna rola semiwariogramów.

### 8.1.2. Metod krigingu

Istnieje szereg meteod krigingu, w tym:

- Kriging prosty (ang. *Simple kriging*)
- Kriging zwykły (ang. *Ordinary kriging*)
- Kriging z trendem (ang. *Kriging with a trend*)
- Kriging danych kodowanych (ang. *Indicator kriging*)
- Kriging stratyfikowany (ang. *Kriging within strata* – KWS)
- Kriging prosty ze zmiennymi średnimi lokalnymi (ang. *Simple kriging with varying local means* - SKlm)
- Kriging z zewnętrznym trendem/Uniwersalny kriging (ang.*Kriging with an external trend/Universal kriging*)
- Kokriging (ang. *Co-kriging*)
- Inne

## 8.2. Kriging prosty

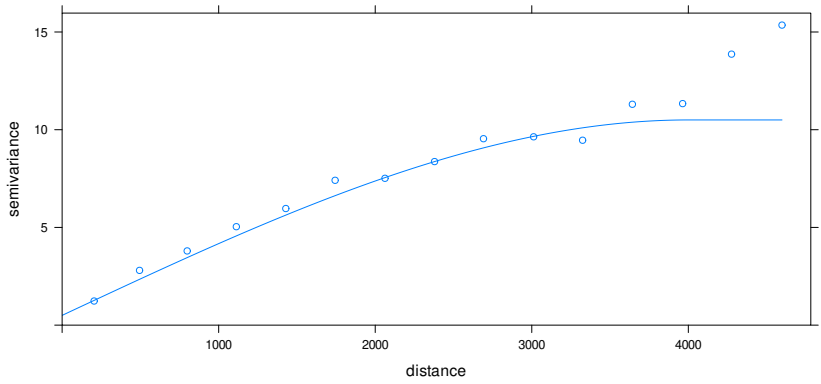
### 8.2.1. Kriging prosty (ang. *Simple kriging*)

Kriging prosty zakłada, że średnia jest znana i stała na całym obszarze. W poniższym przykładzie po stworzeniu semiwariogramu empirycznego, dopasowano model semiwariogramu składający się z funkcji sferycznej o zasięgu 4000 metrów i wartości nuggetu równej 0,5.

```
vario <- variogram(temp~1, locations = punkty)
model <- vgm(10, model = "Sph", range = 4000, nugget = 0.5)
model
```

```
##   model psill range
## 1   Nug   0.5     0
## 2   Sph  10.0  4000
```

```
plot(vario, model = model)
```




```
# fitted <- fit.variogram(vario, model)
```

Następnie nastąpiła estymacja wartości z użyciem metody kriginu prostego. W funkcji `krige()` z pakietu **gstat**, użycie tej metody wymaga ustalenia średniej wartości cechy za pomocą argumentu `beta`.

```
mean(punkty$temp)
```

```
## [1] 15.95036
```

```
sk <- krige(temp~1,
            locations = punkty,
            newdata = siatka,
            model = model,
            beta = 16)
```

```
## [using simple kriging]
```

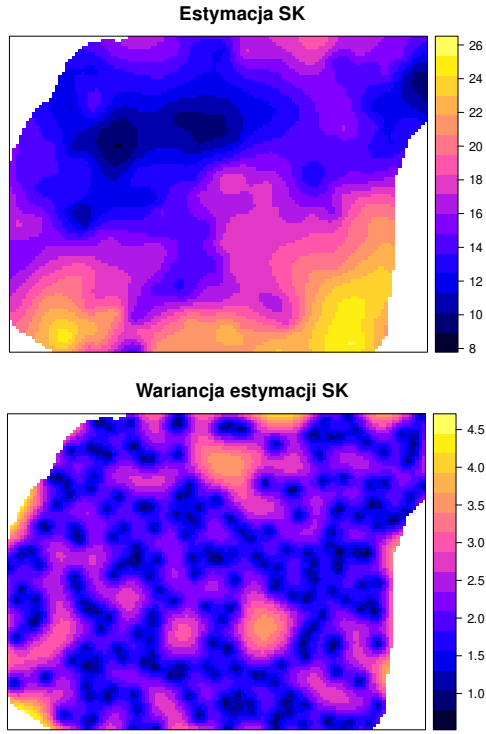
Wynik krigingu prostego, jak i każdy inny uzyskany z użyciem pakietu **gstat**, można podejrzeć używając funkcji `summary()`. Szczególnie ważne są dwie nowe zmienne - `var1.pred` oraz `var1.var`. Pierwsza z nich oznacza wartość estymowaną dla każdego oczka siatki, druga zaś mówi o wariancji estymacji.

```
summary(sk)
```

```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##    var1.pred         var1.var
##  Min.   : 8.934   Min.   :0.7755
##  1st Qu.:13.246   1st Qu.:1.6193
##  Median :15.264   Median :1.9533
##  Mean   :15.877   Mean   :2.0225
##  3rd Qu.:17.900   3rd Qu.:2.3675
##  Max.   :25.369   Max.   :4.4515
```

Obie uzyskane zmienne można wyświetlić z użyciem funkcji `spplot()`.

```
spplot(sk, "var1.pred")
spplot(sk, "var1.var")
```




## 8.3. Kriging zwykły

### 8.3.1. Kriging zwykły (ang. *Ordinary kriging*)

W krigingu zwykłym średnia traktowana jest jako wartość nieznana. Metoda ta uwzględnia lokalne fluktuacje średniej poprzez stosowanie ruchomego okna. Parametry ruchomego okna można określić za pomocą jednego z dwóch argumentów:

- `nmax` - użyta zostanie określona liczba najbliższych obserwacji
- `maxdist` - użyte zostaną jedynie obserwacje w zadanej odległości

```
# ok <- krige(temp~1,
#             locations = punkty,
#             newdata = siatka,
#             model = model,
#             nmax = 30)
ok <- krige(temp~1,
```

```
            locations = punkty,
            newdata = siatka,
            model = model,
            maxdist = 1500)
```

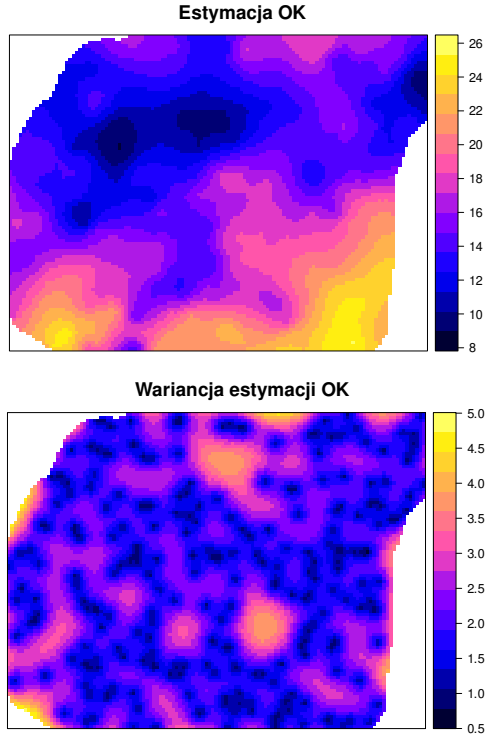
```
## [using ordinary kriging]
```

Podobnie jak w przypadku krigingu prostego, można przyjrzeć się wynikom estymacji używając funkcji `summary()` oraz wyświetlić je używając funkcji `spplot()`.

```
summary(ok)
```

```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##    var1.pred         var1.var
##  Min.   : 8.958   Min.   :0.7757
##  1st Qu.:13.243   1st Qu.:1.6266
##  Median :15.241   Median :1.9691
##  Mean   :15.884   Mean   :2.0562
##  3rd Qu.:18.072   3rd Qu.:2.4076
##  Max.   :25.339   Max.   :4.7311
```

```
spplot(ok, "var1.pred")
spplot(ok, "var1.var")
```

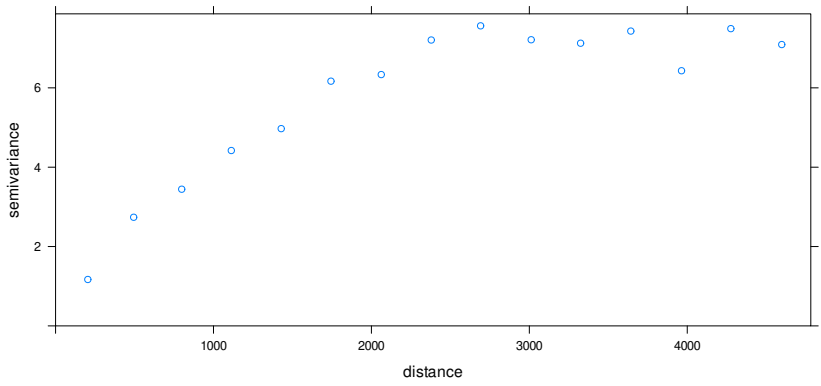



## 8.4. Kriging z trendem

### 8.4.1. Kriging z trendem (ang. *Kriging with a trend*)

Kriging z trendem, określany również jako kriging z wewnętrznym trendem, do estymacji wykorzystuje (oprócz zmienności wartości wraz z odległością) położenie analizowanych punktów. Na poniższym przykładzie w funkcji `variogram()` pierwszy z argumentów przyjął postać `temp~x+y`, co oznacza, że uwzględniamy liniowy trend zależny od współrzędnej `x` oraz `y`.

```
vario_kzt <- variogram(temp~x + y, locations = punkty)
plot(vario_kzt)
```

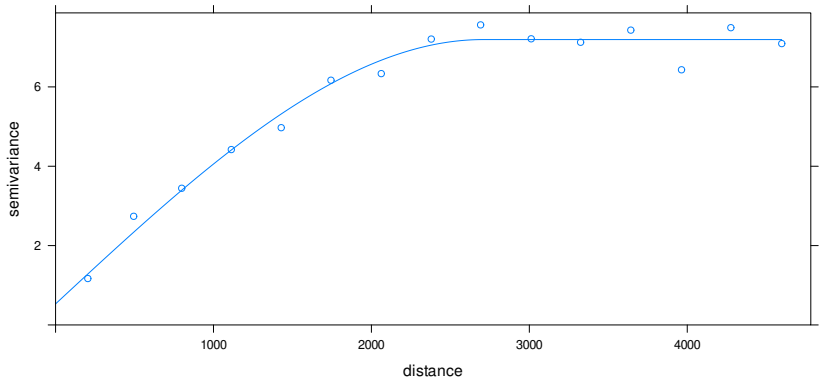



Dalszym etapem jest dopasowanie modelu semiwariancji, a następnie wyliczenie estymowanych wartości z użyciem funkcji `krige()`. Należy tutaj pamiętać, aby wzór (w przykładzie `temp~x + y`) był taki sam podczas budowania semiwariogramu, jak i estymacji.

```
model_kzt <- vgm(model = "Sph", nugget = 1)
fitted_kzt <- fit.variogram(vario_kzt, model_kzt)
fitted_kzt
```

```
##   model     psill    range
## 1   Nug 0.5308661    0.000
## 2   Sph 6.6620797 2705.928
```

```
plot(vario_kzt, fitted_kzt)
```



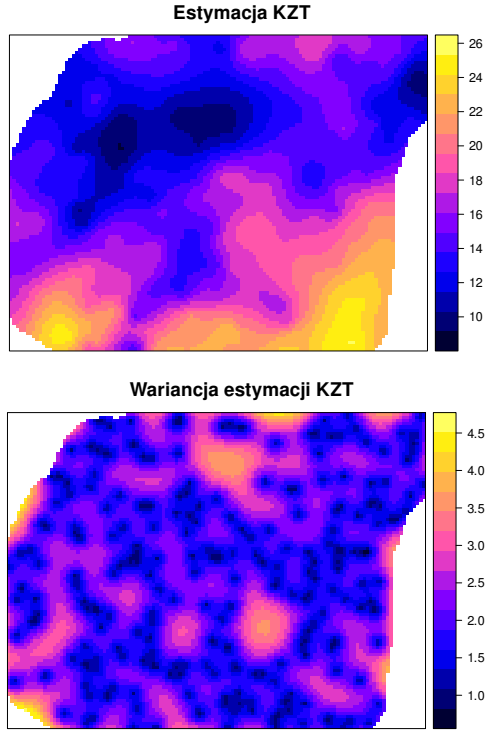
```
kzt <- krige(temp~x + y,
             locations = punkty,
             newdata = siatka,
             model = fitted_kzt)
```

```
## [using universal kriging]
```

```
summary(kzt)
```

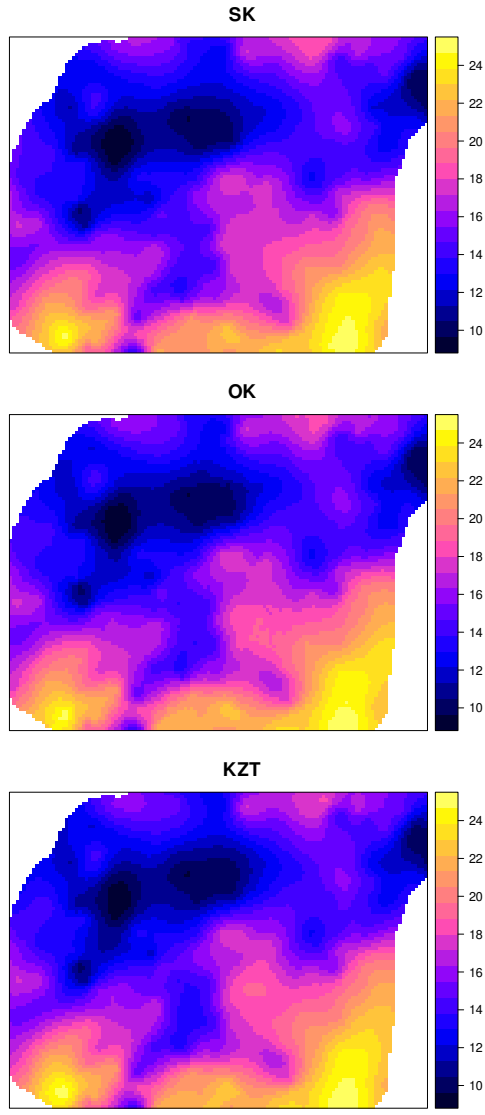
```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##    var1.pred        var1.var
##  Min.   : 9.14   Min.   :0.8141
##  1st Qu.:13.20   1st Qu.:1.6472
##  Median :15.20   Median :1.9749
##  Mean   :15.86   Mean   :2.0481
##  3rd Qu.:18.10   3rd Qu.:2.3847
##  Max.   :25.35   Max.   :4.5134
```

```
spplot(kzt, "var1.pred")
spplot(kzt, "var1.var")
```




## 8.5. Porównanie wyników SK, OK i KZT

Poniższe porównanie krigingu prostego (SK), zwykłego (OK) i z trendem (KZT) wykazuje niewielkie różnice w uzyskanych wynikach. W rozdziałach 9 oraz 10 pokazane będą uzyskane wyniki estymacji temperatury powietrza korzystając z innych metod krigingu.




## 8.6. Zadania

Zadania w tym rozdziale są oparte o dane z obiektu `punkty_pref`. Możesz go wczytać używając poniższego kodu:

```
data(punkty_pref)
```

Na jego podstawie stwórz trzy obiekty - `punkty_pref1` zawierający wszystkie

punkty, `punkty_pref2` zawierający losowe 100 punktów, oraz `punkty_pref3` zawierający losowe 30 punktów.

```
set.seed(2018-11-25)
punkty_pref1 <- punkty_pref
punkty_pref2 <- punkty_pref[sample(nrow(punkty_pref), 100), ]
punkty_pref3 <- punkty_pref[sample(nrow(punkty_pref), 30), ]
```

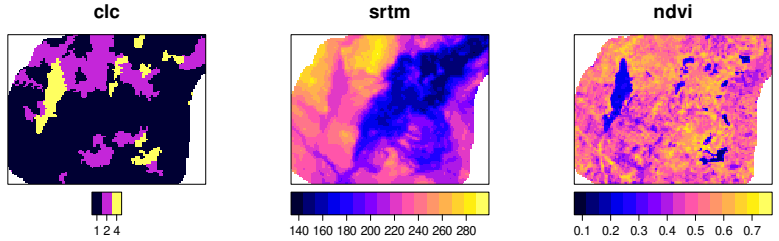
1. Zbuduj optymalne modele semiwariogramu zmiennej `srtm` dla trzech zbiorów danych - `punkty_pref1`, `punkty_pref2`, `punkty_pref3`. Porównaj graficznie uzyskane modele.
2. W oparciu o uzyskane modele stwórz estymacje zmiennej `srtm` dla trzech zbiorów danych - `punkty_pref1`, `punkty_pref2`, `punkty_pref3` używając krigingu prostego. Porównaj graficznie zarówno mapy estymacji jak i mapy wariancji. Opisz zaobserwowane różnice.
3. W oparciu o uzyskane modele stwórz estymacje zmiennej `srtm` dla zbioru danych `punkty_pref3` używając krigingu zwykłego. Sprawdź jak wygląda wynik estymacji uwzględniając (i) 10 najbliższych obserwacji, (ii) 30 najbliższych obserwacji, (iii) obserwacje w odległości do 2 km.
4. Używając krigingu z trendem, stwórz optymalne modele zmiennej `srtm` dla dwóch zbiorów danych - `punkty_pref1` oraz `punkty_pref3`.
5. Porównaj graficznie zarówno mapy estymacji jak i mapy wariancji dla krigingu prostego, zwykłego oraz z trendem dla danych `punkty_pref3`. Jakie można zauważyć podobieństwa a jakie różnice?

# 9. Estymacje używające danych uzupełniających

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(geostatbook)
data(punkty)
data(siatka)
```

W wielu przypadkach, oprócz konkretnych pomiarów, istnieje również informacja na temat zmienności innych cech na analizowanym obszarze. W sytuacji, gdy dodatkowe zmienne są skorelowane ze zmienną analizowaną można wykorzystać jedną z metod krigingu wykorzystującą dane uzupełniające, tj. kriging stratyfikowany, prosty kriging ze zmiennymi średnimi lokalnymi, czy kriging uniwersalny.




## 9.1. Kriging stratyfikowany

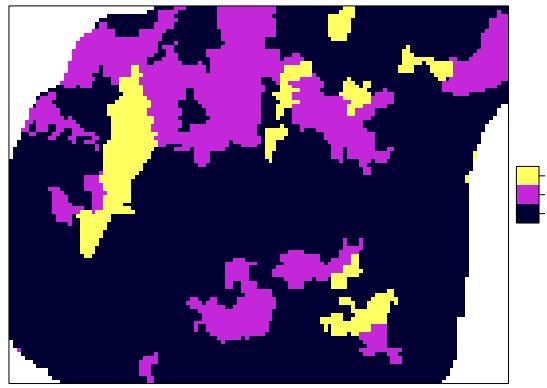
### 9.1.1. Kriging stratyfikowany (ang. *Kriging within strata*)

Kriging stratyfikowany zakłada, że zmienność badanego zjawiska zależy od cechy jakościowej (kategoryzowanej). Przykładowo, wartość badanej

zmiennej jest różna w zależności od pokrycia terenu. Kriging stratyfikowany wymaga posiadania danych zmiennej jakościowej (kategoryzowanej) na całym badanym obszarze.

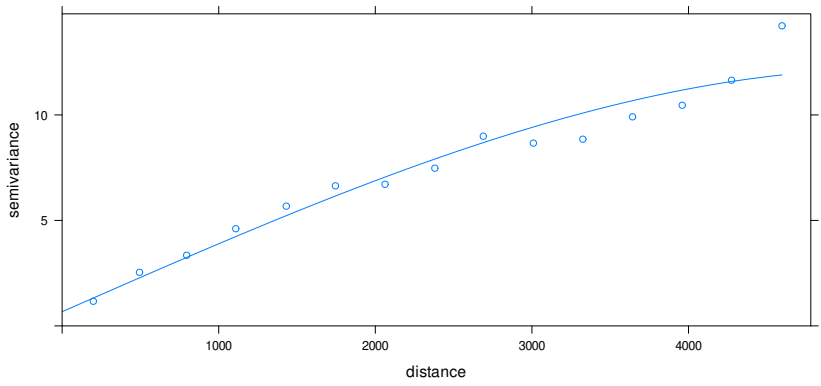
W poniższym przykładzie zmienną jakościową jest uproszczone pokrycie terenu ze zmiennej `clc`. Przyjmuje ono jedno z trzech wartości. `1` oznacza obszary rolnicze, `2` oznacza obszary leśne, a `4` oznacza wody powierzchniowe.

```
siatka$clc <- as.factor(siatka$clc)
spplot(siatka, "clc")
```

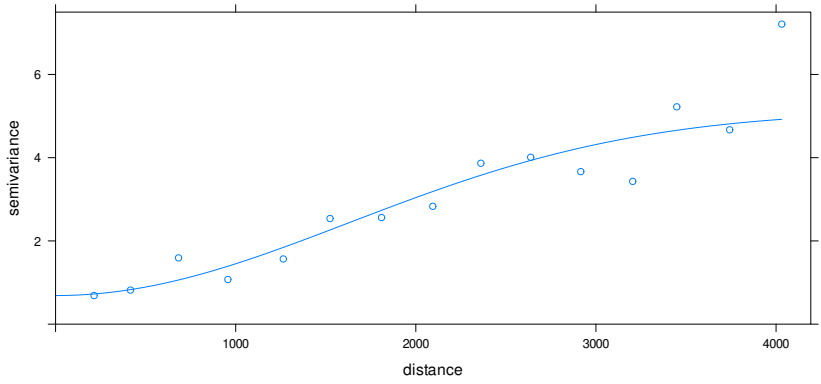



Kriging stratyfikowany polega na niezależnym tworzeniu i modelowaniu semiwariogramów dla każdej z kategorii.

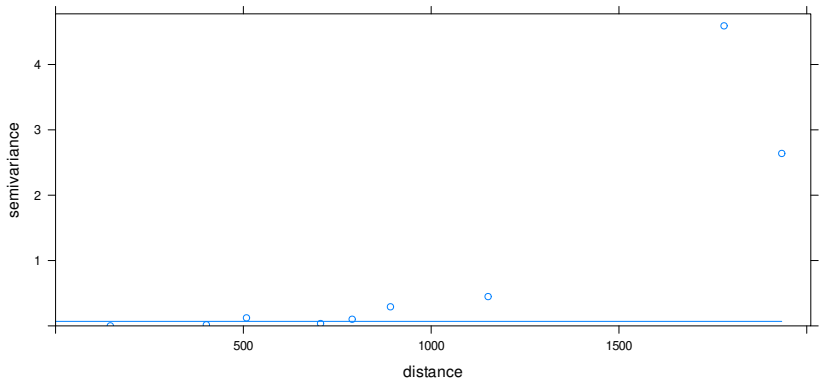
```
vario_kws1 <- variogram(temp~1, punkty[punkty$clc == 1, ])
# plot(vario_kws1)
fitted_kws1 <- fit.variogram(vario_kws1, vgm(model = "Sph", nugget = 0.5))
plot(vario_kws1, fitted_kws1)
```




```
vario_kws2 <- variogram(temp~1, punkty[punkty$clc == 2, ])
# plot(vario_kws2)
fitted_kws2 <- fit.variogram(vario_kws2, vgm(model = "Gau", nugget = 0.1))
plot(vario_kws2, fitted_kws2)
```




```
vario_kws4 <- variogram(temp~1, punkty[punkty$clc == 4, ])
# plot(vario_kws4)
fitted_kws4 <- fit.variogram(vario_kws4, vgm(model = "Nug"))
plot(vario_kws4, fitted_kws4)
```

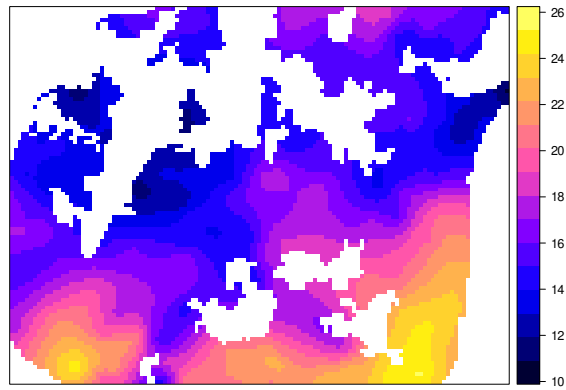



Następnie dla każdego obszaru przeprowadzona jest niezależna estymacja wartości analizowanej cechy. Należy jedynie wcześniej zadbać, by w siatce nie było elementów NA dotyczących zmiennych jakościowych. W przykładzie tworzona jest nowa siatka (siatka2) nie zawierająca braków wartości dla zmiennej clc.

```
siatka2 <- siatka[!is.na(siatka$clc), ]
kws1 <- krige(temp~1,
              location = punkty[punkty$clc == 1, ],
              newdata = siatka2[na.omit(siatka2$clc == 1), ],
              model = fitted_kws1)
```

```
## [using ordinary kriging]
```

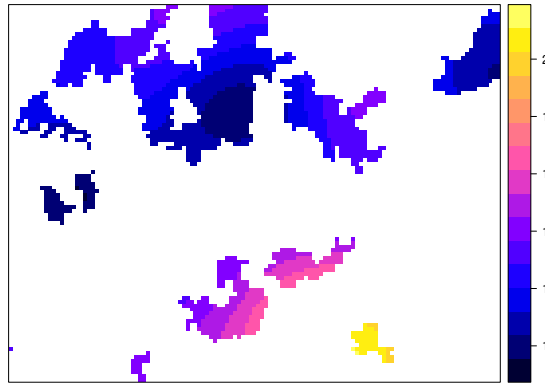
```
spplot(kws1, "var1.pred")
```



```
kws2 <- krige(temp~1,
              location = punkty[punkty$clc == 2, ],
              newdata = siatka2[na.omit(siatka2$clc == 2), ],
              model = fitted_kws2)
```

```
## [using ordinary kriging]
```

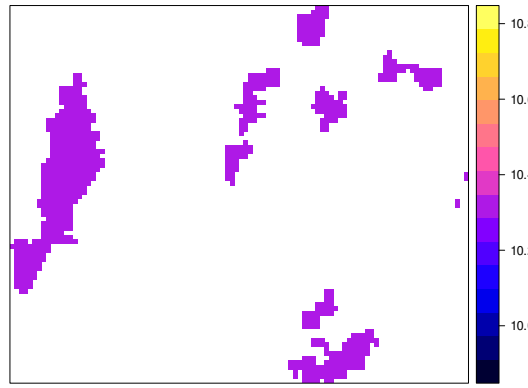
```
spplot(kws2, "var1.pred")
```




```
kws4 <- krige(temp~1,
              location = punkty[punkty$clc == 4, ],
              newdata = siatka2[na.omit(siatka2$clc == 4), ],
              model = fitted_kws4)
```

```
## [using ordinary kriging]
```

```
spplot(kws4, "var1.pred")
```

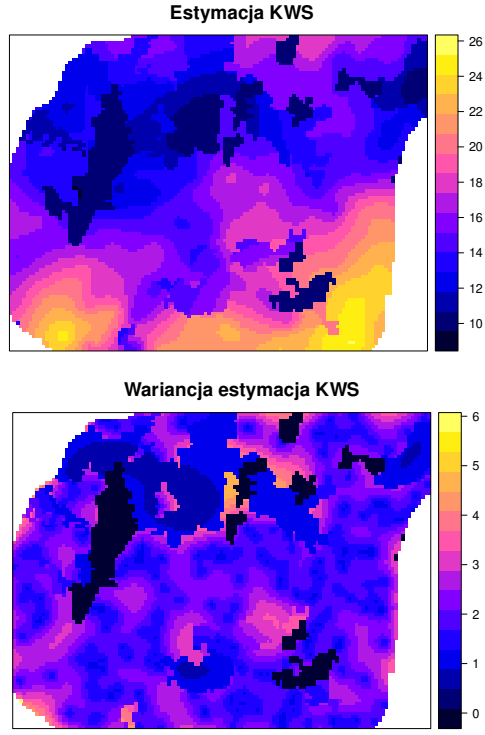



Ostatnim etapem jest połączenie cząstkowych wyników w jeden obiekt klasy `SpatialPixelsDataFrame`.

```
kws <- rbind(kws1, kws2, kws4)
```

Uzyskane w ten sposób wyniki znacząco różnią się od estymacji krigingem prostym czy zwykłym, wykazując odrębność zmienności w poszczególnych kategoriach pokrycia/użytkowania terenu.

```
spplot(kws, "var1.pred")
spplot(kws, "var1.var")
```




## 9.2. Prosty kriging ze zmiennymi średnimi lokalnymi (LVM)

### 9.2.1. Prosty kriging ze zmiennymi średnimi lokalnymi (LVM) (ang. *Simple kriging with varying local means*)

Prosty kriging ze zmiennymi średnimi lokalnymi zamiast znanej (stałej) stacjonarnej średniej wykorzystuje zmienne średnie lokalne uzyskane na podstawie innej informacji.

Lokalna średnia może być uzyskana za pomocą wyliczenia regresji liniowej pomiędzy zmienną badaną a zmienną dodatkową. W takiej sytuacji konieczne jest użycie funkcji `lm()`. W poniższym przykładzie budowany jest model liniowy relacji pomiędzy temperaturą powietrza (`temp`), a wysokością nad poziomem morza (`srtm`).

```
coef <- lm(temp~srtm, punkty)$coef
coef
```

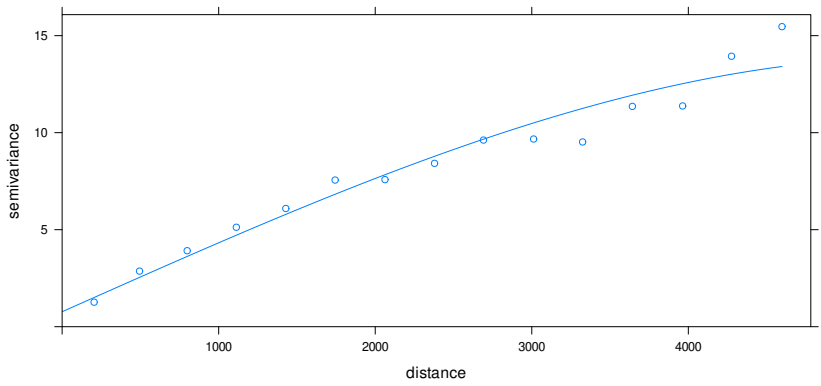
```
##  (Intercept)         srtm
## 17.506469957 -0.007291269
```

Wykorzystując relację pomiędzy tymi dwoma zmiennymi tworzony jest semiwariogram empiryczny, który następnie jest modelowany.

```
vario <- variogram(temp~srtm, location = punkty)
model_sim <- vgm(model = "Sph", nugget = 1)
fitted_sim <- fit.variogram(vario, model_sim)
fitted_sim
```

```
##   model      psill    range
## 1   Nug  0.7705839    0.000
## 2   Sph 13.1155524 5474.628
```

```
plot(vario, model = fitted_sim)
```




Ostatnim krokiem jest estymacja geostatystyczna, w której oprócz czterech podstawowych argumentów, definiujemy także parametr `beta`. W tym wypadku jest to wypadku obiekt uzyskany na podstawie regresji liniowej.

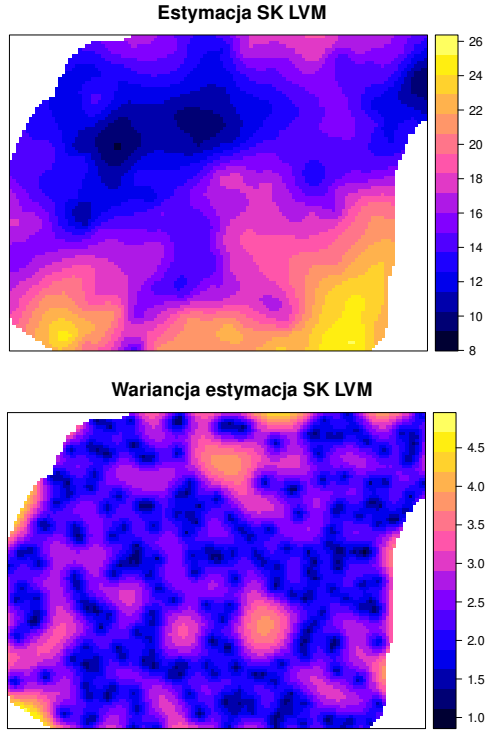
```
sk_lvm <- krige(temp~srtm,
                location = punkty,
                newdata = siatka,
                model = fitted_sim,
                beta = coef)
```

```
## [using simple kriging]
```

```
summary(sk_lvm)
```

```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##    var1.pred         var1.var
##  Min.   : 9.098   Min.   :1.108
##  1st Qu.:13.215   1st Qu.:1.931
##  Median :15.237   Median :2.241
##  Mean   :15.864   Mean   :2.312
##  3rd Qu.:18.049   3rd Qu.:2.637
##  Max.   :25.240   Max.   :4.705
##  NA's   :43       NA's   :43
```

```
spplot(sk_lvm, "var1.pred")
spplot(sk_lvm, "var1.var")
```




## 9.3. Kriging uniwersalny

### 9.3.1. Kriging uniwersalny (ang. *Universal kriging*)

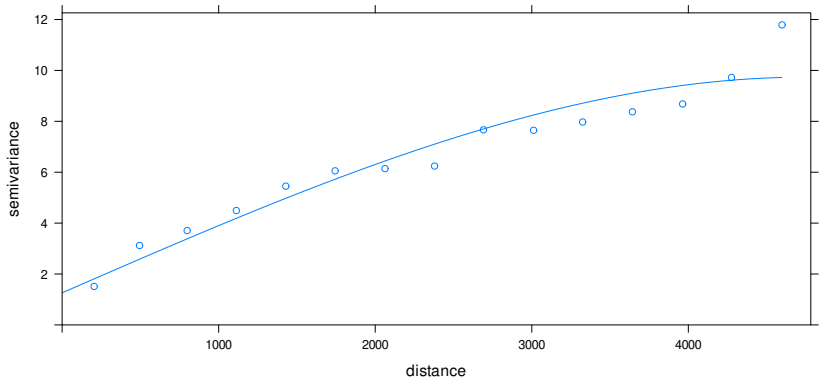
Kriging uniwersalny, określany również jako kriging z trendem (ang. *Kriging with a trend model*) zakłada, że nieznana średnia lokalna zmienia się stopniowo na badanym obszarze. W krigingu uniwersalnym możemy stosować zarówno zmienne jakościowe, jak i ilościowe.

W pierwszym przykładzie, kriging uniwersalny służy stworzeniu semiwariogramu, modelowaniu oraz estymacji temperatury powietrza z użyciem zmiennej pokrycia terenu.

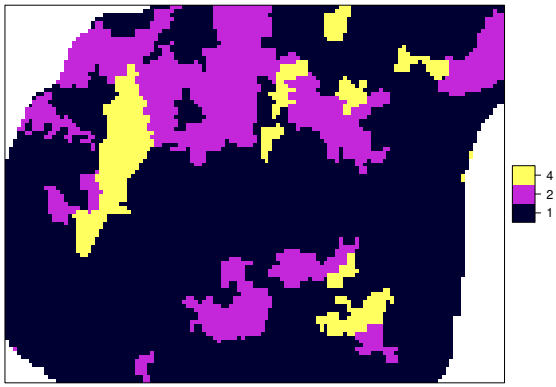
```
punkty$clc <- as.factor(punkty$clc)
vario_uk1 <- variogram(temp~clc, location = punkty)
# plot(vario_uk1)
model_uk1 <- vgm(model = "Sph", nugget = 1)
vario_fit_uk1 <- fit.variogram(vario_uk1, model = model_uk1)
vario_fit_uk1
```

```
##   model    psill    range
## 1   Nug 1.261541    0.000
## 2   Sph 8.472223 4742.211
```

```
plot(vario_uk1, vario_fit_uk1)
```



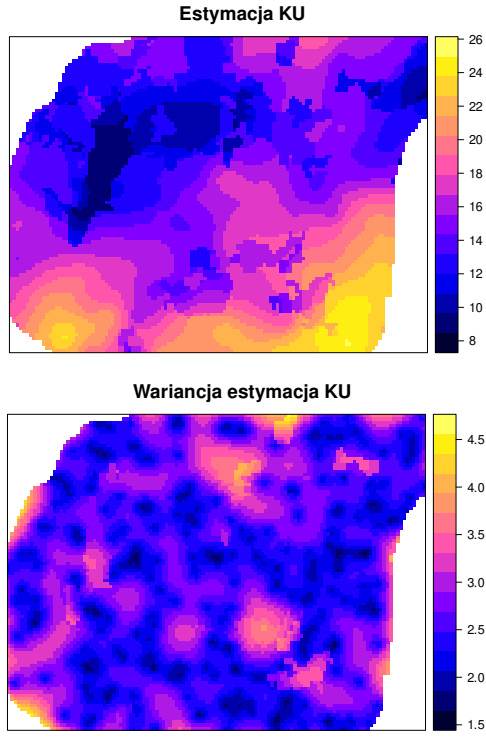

```
siatka$clc <- as.factor(siatka$clc)
spplot(siatka, "clc")
```




```
uk1 <- krige(temp~clc,
             locations = punkty,
             newdata = siatka,
             model = vario_fit_uk1)
```

```
## [using universal kriging]
```

```
spplot(uk1, "var1.pred")
spplot(uk1, "var1.var")
```

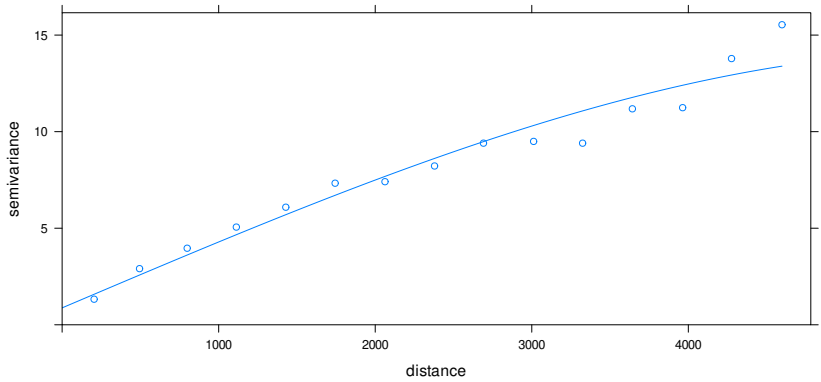



W kolejnym przykładzie zastosowane są już dwie zmienne uzupełniające - wartość wskaźnika wegetacji (`ndvi`) oraz wysokość nad poziomem morza (`srtm`).

```
vario_uk2 <- variogram(temp~ndvi + srtm, location = punkty)
# plot(vario_uk2)
model <- vgm(model = "Sph", nugget = 1)
vario_fit_uk2 <- fit.variogram(vario_uk2, model = model)
vario_fit_uk2
```

```
##   model      psill    range
## 1   Nug  0.8757491    0.000
## 2   Sph 13.3016445 5789.379
```

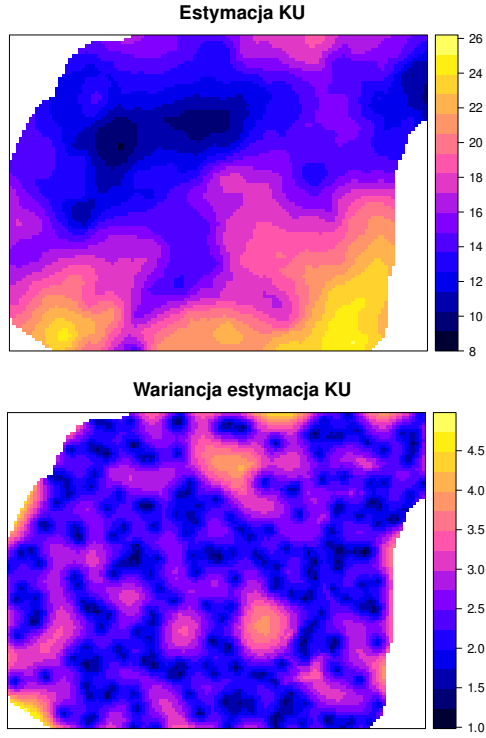
```
plot(vario_uk2, vario_fit_uk2)
```




```
uk2 <- krige(temp~ndvi + srtm,
             locations = punkty,
             newdata = siatka,
             model = vario_fit_uk2)
```

```
## [using universal kriging]
```

```
spplot(uk2, "var1.pred")
spplot(uk2, "var1.var")
```




## 9.4. Zadania

Zadania w tym rozdziale są oparte o dane z obiektu `punkty`.

```
data(punkty)
```

1. Używając krigingu stratyfikowanego, stwórz optymalne modele zmiennej `ndvi` dla trzech typów pokrycia terenu (zmienna `clc`). Bazując na stworzonych modelach, stwórz estymacje zmiennej `ndvi` dla trzech typów pokrycia terenu. Połącz uzyskane estymacje w jedną mapę.
2. Zastosuj prosty kriging ze zmiennymi średnimi lokalnymi do stworzenia estymacji zmiennej `ndvi` używając jej relacji ze zmienną `savi`.
3. Stwórz estymację krigingu uniwersalnego dla zmiennej `ndvi` używając jej relacji ze zmienną `savi`.
4. Stwórz estymację krigingu uniwersalnego dla zmiennej `ndvi` używając jej relacji ze zmiennymi `clc`, `srtm`, `temp` i `savi`.
5. Porównaj graficznie cztery powyższe estymacje. Opisz podobieństwa i różnice.

# 10. Estymacje wielozmienne

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(geostatbook)
data(punkty)
data(punkty_ndvi)
data(siatka)
```

## 10.1. Kokriging

### 10.1.1. Kokriging (ang. *co-kriging*)

Kokriging pozwala na wykorzystanie dodatkowej zmiennej (ang. *auxiliary variable*), zwanej inaczej kozmienną (ang. *co-variable*), która może być użyta do prognozowania wartości badanej zmiennej w nieopróbowanej lokalizacji. Zmienna dodatkowa może być pomierzona w tych samych miejscach, gdzie badana zmienna, jak też w innych niż badana zmienna. Możliwa jest też sytuacja, gdy zmienna dodatkowa jest pomierzona w dwóch powyższych przypadkach. Kokriging wymaga, aby obie zmienne były istotnie ze sobą skorelowane. Najczęściej kokriging jest stosowany w sytuacji, gdy zmienna dodatkowa jest łatwiejsza (tańsza) do pomierzenia niż zmienna główna. W efekcie, uzyskany zbiór danych zawiera informacje o badanej zmiennej oraz gęściej opróbowane informacje o zmiennej dodatkowej. Jeżeli informacje o zmiennej dodatkowej są znane dla całego obszaru wówczas bardziej odpowiednią techniką będzie kriging z zewnętrznym trendem (KED).

### 10.1.2. Wybór dodatkowej zmiennej

Wybór zmiennej dodatkowej może opierać się na dwóch kryteriach:

- Teoretycznym
- Empirycznym

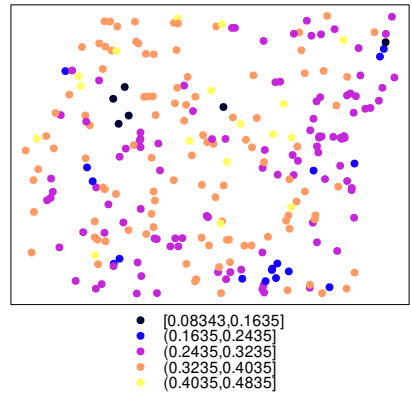
## 10.2. Krossemiwariogramy

### 10.2.1. Krossemiwariogramy (ang. crossvariogram)

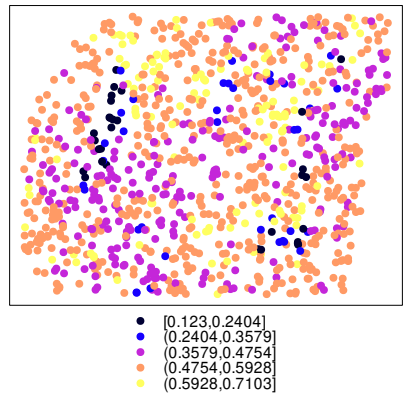
Metoda kokrigingu opiera się nie o semiwariogram, lecz o krossemiwariogramy. Krossemiwariogram jest to wariancja różnicy pomiędzy dwiema zmiennymi w dwóch lokalizacjach. Wyliczając krossemiwariogram otrzymujemy empiryczne semiwariogramy dla dwóch badanych zmiennych oraz krosswariogram dla kombinacji dwóch zmiennych.

W poniższym przykładzie istnieją dwie zmienne, `savi` ze zbioru `punkty` pomierzona w 250 lokalizacjach oraz `ndvi` ze zbioru `punkty_ndvi` pomierzona w 997 punktach.

```
spplot(punkty, "savi")
```




```
spplot(punkty_ndvi, "ndvi")
```



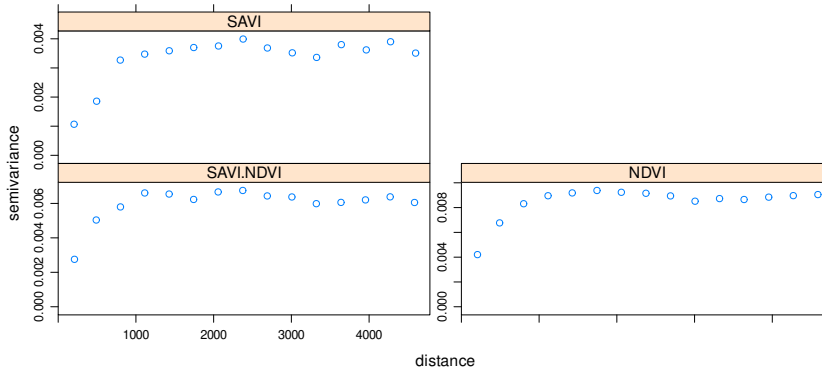

Tworzenie krossemiwariogramów odbywa się z użyciem funkcji `gstat()`. Na początku definiujemy pierwszy obiekt `g`. Składa się on z obiektu pustego (`NULL`), nazwy pierwszej zmiennej (nazwa może być dowolna), wzoru (w przykładzie `savi~1`), oraz pierwszego zbioru punktowego. Następnie do pierwszego obiektu `g` dodajemy nowe informacje również poprzez funkcję `gstat()`. Jest to nazwa obiektu (`g`), nazwa drugiej zmiennej, wzór, oraz drugi zbiór punktowy.

```
g <- gstat(NULL,
           id = "SAVI",
           form = savi~1,
           data = punkty)
g <- gstat(g,
           id = "NDVI",
           form = ndvi~1,
           data = punkty_ndvi)
g
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 250 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 997 x 1
```

Z uzyskanego w ten sposób obiektu tworzymy krossemiwariogram (funkcja `variogram()`), a następnie go wizualizujemy używając funkcji `plot()`.

```
v <- variogram(g)
plot(v)
```




## 10.3. Modelowanie krossemiwariogramów

Modelowanie krossemiwariogramów, podobnie jak ich tworzenie, odbywa się używając funkcji `gstat()`. Podaje się w niej wcześniejszy obiekt `g`, model, oraz argument `fill.all = TRUE`. Ten ostatni parametr powoduje, że model dodawany jest do wszystkich elementów krossemiwariogramu.

```
g_model <- vgm(0.006, model = "Sph",
               range = 2000, nugget = 0.001)
g <- gstat(g, model = g_model, fill.all = TRUE)
g
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 250 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 997 x 1
## variograms:
##               model psill range
## SAVI[1]         Nug 0.001     0
## SAVI[2]         Sph 0.006  2000
## NDVI[1]         Nug 0.001     0
## NDVI[2]         Sph 0.006  2000
## SAVI.NDVI[1]    Nug 0.001     0
## SAVI.NDVI[2]    Sph 0.006  2000
```

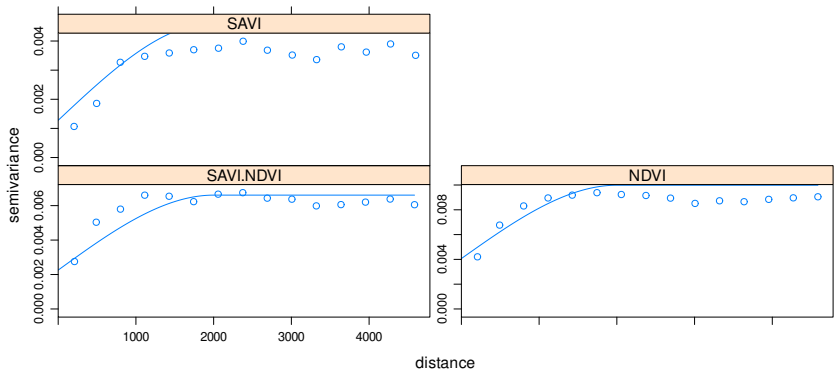
W przypadku semiwariogramów funkcja `fit.variogram()` służyła dopasowaniu parametrów modelu do semiwariogramu empirycznego. Podobną rolę w krossemiwariogramach spełnia funkcja `fit.lmc()`. Dopasowuje ona liniowy model koregionalizacji do semiwariogramów wielozmienych. Funkcja `fit.lmc()` oczekuje co najmniej dwóch elementów, krossemiwariogra-

mu oraz modelów krossemiwariancji. W poniższym przykładzie dodatkowo użyto parametru `correct.diagonal = 1.01`, z uwagi na to że analizowane zmienne wykazywały bardzo silną korelację.

```
g_fit <- fit.lmc(v, g, correct.diagonal = 1.01)
g_fit
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 250 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 997 x 1
## variograms:
##               model       psill range
## SAVI[1]         Nug 0.001278509     0
## SAVI[2]         Sph 0.003338946  2000
## NDVI[1]         Nug 0.004065525     0
## NDVI[2]         Sph 0.005918453  2000
## SAVI.NDVI[1]    Nug 0.002257298     0
## SAVI.NDVI[2]    Sph 0.004352821  2000
```

```
plot(v, g_fit)
```




## 10.4. Kokriging

Posiadając dopasowane modele oraz siatkę można uzyskać wynik używając funkcji `predict()`.

```
ck <- predict(g_fit, newdata = siatka)
```

```
## Linear Model of Coregionalization found. Good.
## [using ordinary cokriging]
```

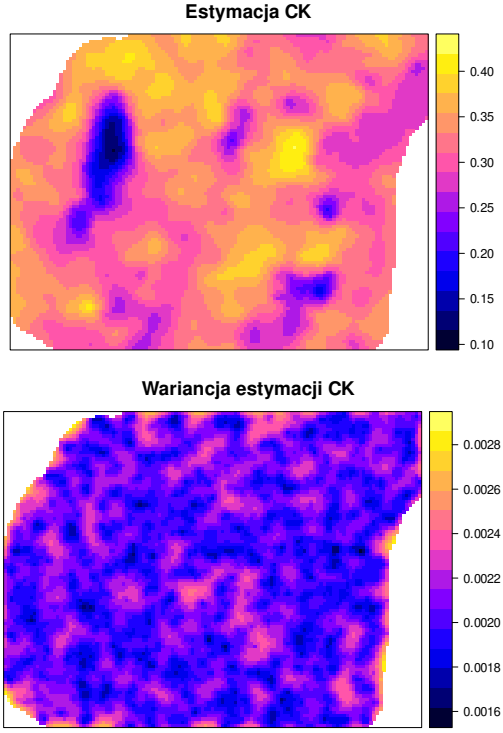
W efekcie otrzymujemy pięć zmiennych:

1. `SAVI.pred` - estymacja zmiennej `savi`
2. `SAVI.var` - wariancja zmiennej `savi`
3. `NDVI.pred` - estymacja zmiennej `ndvi`
4. `NDVI.var` - wariancje zmiennej `ndvi`
5. `cov.SAVI.NDVI` - kowariancje zmiennych `savi` oraz `ndvi`

```
summary(ck)
```

```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##    SAVI.pred         SAVI.var           NDVI.pred         NDVI.var
##  Min.   :0.1152   Min.   :0.001615   Min.   :0.2336   Min.   :0.004750
##  1st Qu.:0.3001   1st Qu.:0.001958   1st Qu.:0.4715   1st Qu.:0.005188
##  Median :0.3255   Median :0.002046   Median :0.5056   Median :0.005336
##  Mean   :0.3205   Mean   :0.002060   Mean   :0.4996   Mean   :0.005365
##  3rd Qu.:0.3480   3rd Qu.:0.002149   3rd Qu.:0.5365   3rd Qu.:0.005513
##  Max.   :0.4197   Max.   :0.002862   Max.   :0.6280   Max.   :0.006780
##  cov.SAVI.NDVI
##  Min.   :0.002658
##  1st Qu.:0.003045
##  Median :0.003157
##  Mean   :0.003178
##  3rd Qu.:0.003289
##  Max.   :0.004232
```

```
spplot(ck, "SAVI.pred")
spplot(ck, "SAVI.var")
```




## 10.5. Zadania

Zadania w tym rozdziale są oparte o dane z `meuse` z pakietu **sp**.

```
library(sp)
data("meuse")
coordinates(meuse) = ~x + y
```

Na jego podstawie wydziel dwa obiekty - `meuse155` zawierający tylko zmienną `lead` dla 155 punktów, oraz `meuse60` zawierający tylko zmienną `copper` dla 60 punktów.

```
meuse155 <- meuse["lead"]
meuse60 <- meuse[sample(nrow(meuse), 60), "copper"]
```

1. Stwórz siatkę interpolacyjną o rozdzielczości 100 jednostek dla obszaru, w którym znajdują się punkty `meuse`.
2. Zbuduj optymalne modele semiwariogramu zmiennej `lead` dla obiektu `meuse155` oraz zmiennej `copper` dla obiektu `meuse60`. Porównaj graficznie uzyskane modele.
3. Korzystając z obiektów `meuse155` oraz `meuse60` stwórz krossemiwariogram.
4. Zbuduj ręczny model uzyskanego krossemiwariogramu. Następnie stwórz model automatyczny. Porównaj uzyskane wyniki.
5. Stwórz estymację zmiennej `copper` w nowo utworzonej siatce korzystając z kokrigingu.

# 11. Estymacja lokalnego rozkładu prawdopodobieństwa

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
```

## 11.1. Kriging danych kodowanych

### 11.1.1. Kriging danych kodowanych (ang. *Indicator kriging*)

Kriging danych kodowanych to metoda krigingu oparta o dane kategoryzowane lub też dane przetworzone z postaci ciągłej do binarnej. Jest ona zazwyczaj używana jest to oszacowania prawdopodobieństwa przekroczenia zdefiniowanej wartości progowej, może być również używana do szacowania wartości z całego rozkładu. Wartości danych wykorzystywane do krigingu danych kodowanych są określone jako 0 lub 1, co reprezentuje czy wartość danej zmiennej jest powyżej czy poniżej określonego progu.

### 11.1.2. Wady i zalety krigingu danych kodowanych

Zalety:

- Możliwość zastosowania, gdy nie interesuje nas konkretna wartość, ale znalezienie obszarów o wartości przekraczającej dany poziom
- Nie jest istotny kształt rozkładu

Wady:

- Potencjalnie trudne do modelowania semiwariogramy (szczególnie skrajnych przedziałów)
- Czasochłonność/pracochłonność

## 11.2. Przykłady krigingu danych kodowanych

### 11.2.1. Binaryzacja danych

Pierwszym krokiem w krigingu danych kodowanych jest stworzenie zmiennej binarnej. Na poniższym przykładzie tworzona jest nowa zmienna `temp_ind`. Przyjmuje ona wartość `TRUE` (czyli `1`) dla pomiarów temperatury wyższych niż 20 stopni Celsjusza, a dla pomiarów równych i niższych niż 20 stopni Celsjusza jej wartość wynosi `FALSE` (czyli `0`).

```
summary(punkty$temp)
```

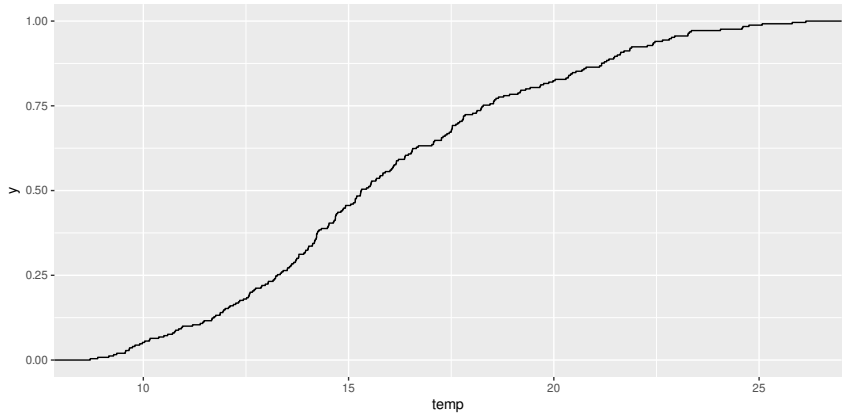
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   8.706  13.284  15.309  15.950  18.273  26.139
```

```
punkty$temp_ind <- punkty$temp > 20
summary(punkty$temp_ind)
```

```
##    Mode   FALSE    TRUE
## logical     206      44
```

W przykładzie, próg został wyznaczony arbitralnie. Istnieje oczywiście szereg innych możliwości wyznaczania progu. Można wykorzystać wiedzę zewnętrzną (np. toksyczne stężenie analizowanej substancji) lub też posłużyć się wykresem dystrybuanty do określenia istotnej zmiany wartości.

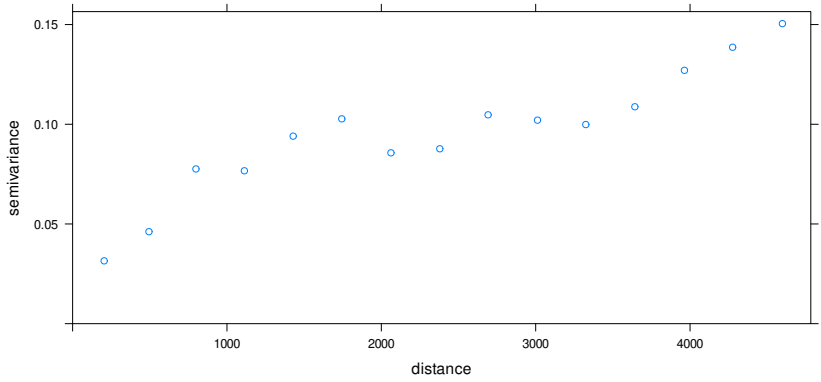
```
ggplot(punkty@data, aes(temp)) + stat_ecdf()
```




## 11.2.2. Modelowanie

Tworzenie i modelowanie semiwariogramu empirycznego w krigingu danych kodowanych wygląda tak samo jak, np. w przypadku krigingu zwykłego. Korzystając z funkcji `variogram()` tworzony jest semiwariogram empiryczny, używając `vgm()` tworzony jest model "ręczny", który następnie jest dopasowywany z użyciem funkcji `fit.variogram()`.

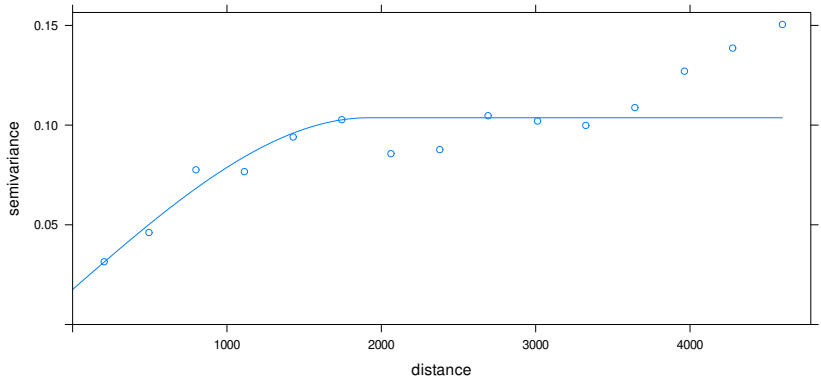
```
vario_ind <- variogram(temp_ind~1, locations = punkty)
plot(vario_ind)
```




```
model_ind <- vgm(model = "Sph", nugget = 0.01)
fitted_ind <- fit.variogram(vario_ind, model_ind)
fitted_ind
```

```
##   model      psill    range
## 1   Nug 0.01761350    0.000
## 2   Sph 0.08607693 1919.258
```

```
plot(vario_ind, model = fitted_ind)
```




### 11.2.3. Estymacja

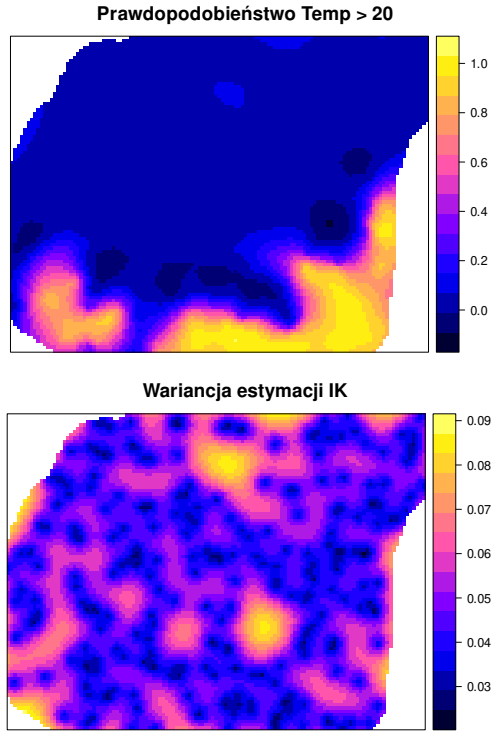
Ostatnim etapem jest stworzenie interpolacji geostatystycznej z pomocą funkcji `krige`. Wymaga ona czterech argumentów - wzoru (`temp_ind~1`), zbioru punktowego (`punkty`), siatki do interpolacji (`siatka`) oraz modelu (`fitted_ind`).

```
ik <- krige(temp_ind~1,
            locations = punkty,
            newdata = siatka,
            model = fitted_ind)
```

```
## [using ordinary kriging]
```

W wyniku estymacji otrzymuje się mapę przestawiającą prawdopodobieństwo przekroczenia zadanej wartości (w tym wypadku jest to 20 stopni Celsjusza) oraz uzyskaną wariancję estymacji.

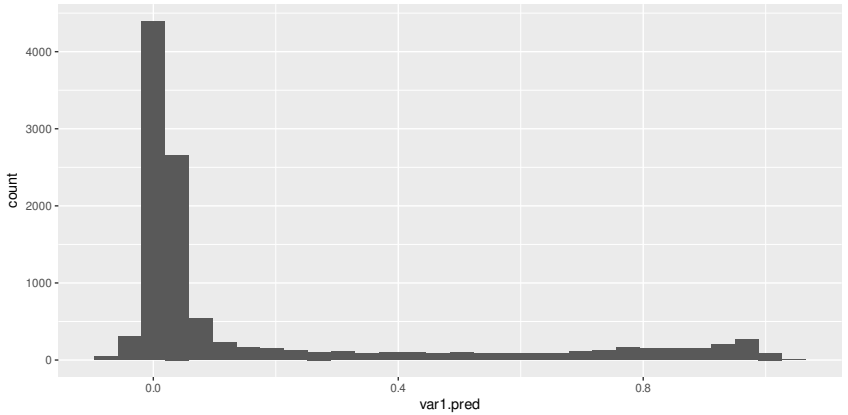
```
spplot(ik, "var1.pred")
spplot(ik, "var1.var")
```




### 11.2.4. Tworzenie mapy binarnej

Mapy przestawiające prawdopodobieństwo można też przetworzyć do postaci map binarnych poprzez użycie wybranej wartości progowej. Rozkład wartości prawdopodobieństwa jest możliwy do zobaczenia, np. używając histogramu:

```
ggplot(ik@data, aes(var1.pred)) + geom_histogram()
```

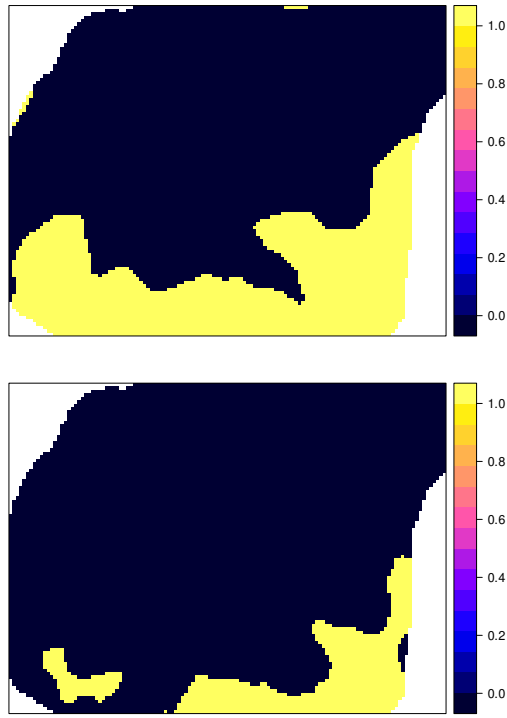



Następnie można stworzyć nową zmienną binarną w oparciu o wartości estymacji. Poniższy kod tworzy dwie nowe zmienne - `prog1`, gdzie wartość progowa została arbitralnie ustalona na 0.1 oraz `prog2` z wartością progową 0.75.

```
ik$prog1 <- ik$var1.pred > 0.1
ik$prog2 <- ik$var1.pred > 0.75
```

W wyniku otrzymuje się binarne mapy, dla których stwierdza się obszary powyżej lub poniższej zadanego prawdopodobieństwa.

```
spplot(ik, "prog1")
spplot(ik, "prog2")
```




### 11.2.5. Alternatywne użycie funkcji

Alternatywnie, zamiast tworzenia nowej zmiennej (takiej jak `temp_ind`), można wykorzystać funkcję `I`. Z jej użyciem można definiować przyjęte progi bezpośrednio do funkcji `variogram` i `krige`. Na poniższych przykładach w ten sposób ustalono trzy progi - poniżej 20, poniżej 16, oraz poniżej 12 stopni Celsjusza.

```
vario_ind20 <- variogram(I(temp < 20)~1, locations = punkty)
fitted_ind20 <- fit.variogram(vario_ind20,
                              vgm("Sph", nugget = 0.01))
vario_ind16 <- variogram(I(temp < 16)~1, locations = punkty)
fitted_ind16 <- fit.variogram(vario_ind16,
                              vgm("Sph", nugget = 0.03))
vario_ind12 <- variogram(I(temp < 12)~1, locations = punkty)
fitted_ind12 <- fit.variogram(vario_ind12,
                              vgm("Sph", nugget = 0.03))
```

```
ik20 <- krige(I(temp < 20)~1,
              locations = punkty,
              newdata = siatka,
              model = fitted_ind20,
              nmax = 30)
```

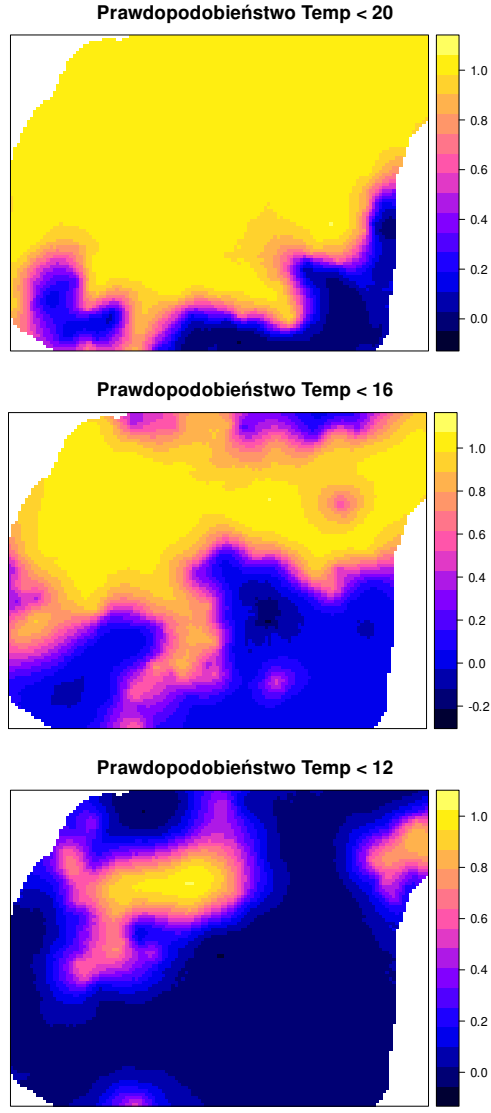
```
## [using ordinary kriging]
```

```
ik16 <- krige(I(temp < 16)~1,
              locations = punkty,
              newdata = siatka,
              model = fitted_ind16,
              nmax = 30)
```

```
## [using ordinary kriging]
```

```
ik12 <- krige(I(temp < 12)~1,
              locations = punkty,
              newdata = siatka,
              model = fitted_ind12,
              nmax = 30)
```

```
## [using ordinary kriging]
```




## 11.3. Zadania

Zadania w tym rozdziale są oparte o dane z obiektu `punkty_ndvi`.

```
data(punkty_ndvi)
```

1. Używając obiektu `punkty_ndvi` stwórz nowe zmienne określające czy zmienna `ndvi` ma wartość:

- poniżej 0.3
- pomiędzy 0.3 a 0.5
- powyżej 0.5

2. Poczytaj na temat tego wskaźnika. Co mogą oznaczać powyższe przedziały?
3. Korzystając z trzech powyższych przedziałów stwórz mapy prawdopodobieństwa. W jaki sposób można zinterpretować trzy uzyskane mapy?
4. Używając wybranego progu prawdopodobieństwa, stwórz trzy mapy binarne.
5. (Dodatkowe) połącz trzy mapy binarne w jedną mapę pokazującą uproszczone pokrycie terenu.

# 12. Ocena jakości estymacji

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

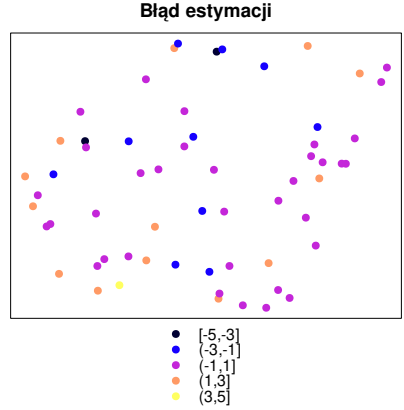
```
library(sp)
library(gstat)
library(caret)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
```

## 12.1. Wizualizacja jakości estymacji

### 12.1.1. Mapa

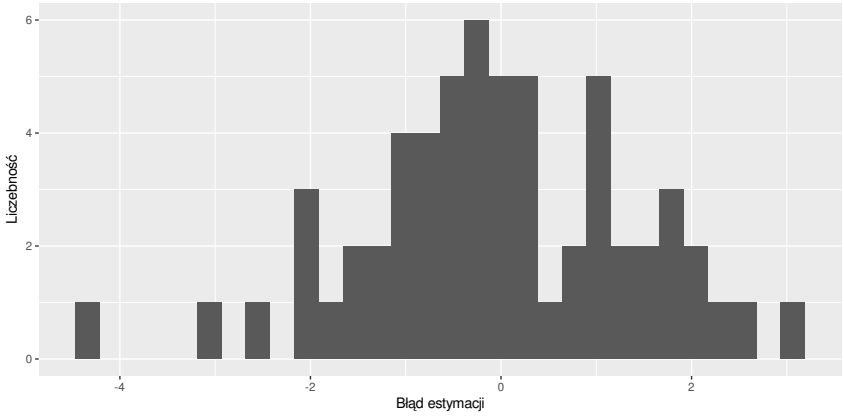
Do oceny przestrzennej jakości estymacji można zastosować mapę przestawiającą błędy estymacji. Wyliczenie błędów estymacji odbywa się poprzez odjęcie od estymacji wartości obserwowanej.

```
blad_estymacji <- obserwowane - estymacja
```



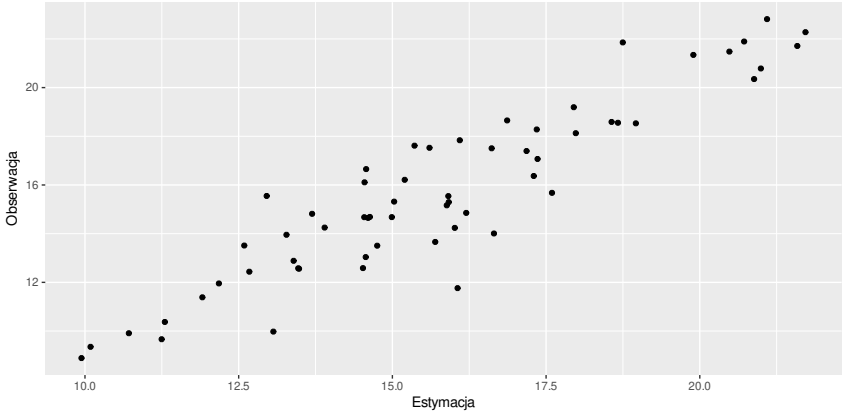
### 12.1.2. Histogram

Błąd estymacji można również przedstawić na wykresach, między innymi, na histogramie.




### 12.1.3. Wykres rozrzutu

Do porównania pomiędzy wartością estymowaną a obserwowaną może również posłużyć wykres rozrzutu.



## 12.2. Statystyki jakości estymacji

### 12.2.1. Podstawowe statystyki

W momencie, gdy trzeba określić jakość estymacji lub porównać wyniki pomiędzy estymacjami należy zastosować tzw. statystyki jakości estymacji. Do podstawowych statystyk ocen jakości estymacji należą:

- Średni błąd estymacji (MPE, ang. *mean prediction error*)
- Pierwiastek średniego błędu kwadratowego (RMSE, ang. *root mean square error*)
- Współczynnik determinacji ($R^2$, ang. *coefficient of determination*)

Idealna estymacja dawałaby brak błędu oraz współczynnik determinacji pomiędzy pomiarami (całą populacją) i szacunkiem równy 1. Należy jednak zdawać sobie sprawę, że dane wejściowe są obarczone błędem/niepewnością, dlatego też w praktyce idealna estymacja nie jest osiągana. Wysokie, pojedyncze wartości błędu mogą świadczyć, np. o wystąpieniu wartości odstających.

### 12.2.2. Średni błąd estymacji

Średni błąd estymacji (MPE) można wyliczyć korzystając z poniższego wzoru:

$$MPE = \frac{\sum_{i=1}^{n}(v_i - \hat{v}_i)}{n}$$

, gdzie $v_i$ to wartość obserwowana a $\hat{v}_i$ to wartość estymowana.

Średni błąd estymacji można też wyliczyć używając funkcji `mean()` w R.

```
MPE <- mean(obserwowane - estymacja)
```

Optymalnie wartość średniego błędu estymacji powinna być jak najbliżej 0.

### 12.2.3. Pierwiastek średniego błędu kwadratowego

Pierwiastek średniego błędu kwadratowego (RMSE) jest możliwy do wyliczenia poprzez wzór:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(v_i - \hat{v}_i)^2}{n}}$$

, gdzie $v_i$ to wartość obserwowana a $\hat{v}_i$ to wartość estymowana.

RMSE można też wyliczyć w R.

```
RMSE <- sqrt(mean((obserwowane - estymacja) ^ 2))
```

Optymalnie wartość pierwiastka średniego błędu kwadratowego powinna być jak najmniejsza.

### 12.2.4. Współczynnik determinacji

Współczynnik determinacji ($R^2$) jest możliwy do wyliczenia poprzez wzór:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{v}_i - v_i)^2}{\sum_{i=1}^{n}(v_i - \overline{v_i})^2}$$

, gdzie $v_i$ to wartość obserwowana, $\hat{v}_i$ to wartość estymowana, a $\overline{v}$ średnia arytmetyczna wartości obserwowanych.

$R^2$ można też wyliczyć w R.

```
R2 <- 1 - sum((estymacja - obserwowane) ^ 2) /
    sum((obserwowane - mean(obserwowane)) ^ 2)
```

lub

```
R2 <- cor(obserwowane, estymacja) ^ 2
```

Współczynnik determinacji przyjmuje wartości od 0 do 1, gdzie model jest lepszy im wartość tego współczynnika jest bliższa jedności.

## 12.3. Jakość wyników estymacji

### 12.3.1. Walidacja wyników estymacji

Dokładne dopasowanie modelu do danych może w efekcie nie dawać najlepszych wyników. Szczególnie będzie to widoczne w przypadku modelowania, w którym dane obarczone są znacznym szumem (zawierają wyraźny błąd) lub też posiadają kilka wartości odstających. W efekcie ważne jest stosowanie metod pozwalających na wybranie optymalnego modelu. Do takich metod należy, między innymi, walidacja podzbiorem (ang. *jackknifing*) oraz kroswalidacja (ang. *crossvalidation*).

### 12.3.2. Walidacja podzbiorem

Walidacja podzbiorem polega na podziale zbioru danych na dwa podzbiory - treningowy i testowy. Zbiór treningowy służy do stworzenia semiwariogramu empirycznego, zbudowania modelu oraz estymacji wartości. Następnie wynik estymacji porównywany jest z rzeczywistymi wartościami ze zbioru testowego. Zaletą tego podejścia jest stosowanie danych niezależnych od estymacji do oceny jakości modelu. Wadą natomiast jest konieczność posiadania (relatywnie) dużego zbioru danych.

Na poniższym przykładzie zbiór danych dzielony jest używając funkcji `createDataPartition()` z pakietu **caret**. Użycie tej funkcji powoduje stworzenie indeksu zawierającego numery wierszy dla zbioru treningowego. Ważną zaletą funkcji `createDataPartition()` jest to, iż w zbiorze treningowym i testowym zachowane są podobne rozkłady wartości. W przykładzie użyto argumentu `p = 0.75`, który oznacza, że 75% danych będzie należało do zbioru treningowego, a 25% do zbioru testowego. Następnie korzystając ze stworzonego indeksu, budowane są dwa zbiory danych - treningowy (`train`) oraz testowy (`test`).

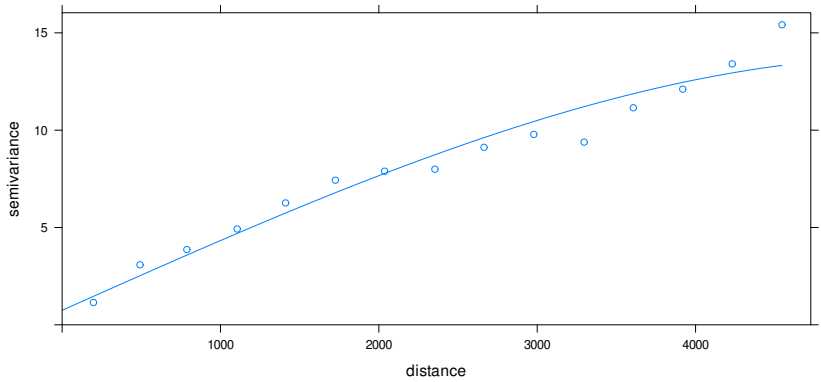
```
set.seed(124)
indeks <- as.vector(createDataPartition(punkty$temp,
                                        p = 0.75,
                                        list = FALSE))
indeks
```

```
##   [1]   1   2   4   6   7   8   9  10  11  12  13  14  17  18  19  20  21  22
##  [19]  27  28  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
##  [37]  46  47  48  49  50  53  54  56  57  58  59  60  61  62  64  65  66  67
##  [55]  69  70  71  74  75  76  78  79  80  82  83  84  86  87  89  90  91  93
##  [73]  94  95  97  98  99 100 101 103 106 110 111 112 113 114 116 117 118 119
##  [91] 120 121 122 123 124 125 126 127 128 129 131 132 133 135 137 138 139 140
## [109] 141 142 143 144 145 146 150 153 154 155 157 160 162 164 165 167 168 169
## [127] 170 171 172 173 174 175 176 177 178 181 182 183 184 185 186 187 188 189
## [145] 191 195 196 197 198 199 200 201 203 204 205 206 207 208 209 210 212 213
## [163] 215 217 218 219 221 223 224 225 226 227 228 229 230 231 232 233 235 236
## [181] 237 239 240 241 242 243 244 248 249 250
```

```
train <- punkty[indeks, ]
test <- punkty[-indeks, ]
```

Dalszym krokiem jest stworzenie semiwariogramu empirycznego oraz jego modelowanie w oparciu o zbiór treningowy.

```
vario <- variogram(temp~1, locations = train)
model <- vgm(model = "Sph", nugget = 0.5)
fitted <- fit.variogram(vario, model)
plot(vario, model = fitted)
```




Do porównania wyników estymacji w stosunku do zbioru testowego posłuży funkcja `krige()`. Wcześniej wymagała ona podania wzoru, zbioru punktowego, siatki oraz modelu. W tym przypadku jednak chcemy porównać wynik estymacji i testowy zbiór punktowy. Dlatego też, zamiast obiektu siatki definiujemy obiekt zawierający zbiór testowy (`test`).

```
test_sk <- krige(temp~1,
                 locations = train,
                 newdata = test,
                 model = fitted,
                 beta = 16)
```

```
## [using simple kriging]
```

```
summary(test_sk)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## x 745546.9 756666.7
## y 712999.4 721118.7
## Is projected: TRUE
## proj4string :
```

```
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 60
## Data attributes:
##    var1.pred         var1.var
##  Min.   : 9.943   Min.   :1.315
##  1st Qu.:13.641   1st Qu.:2.042
##  Median :15.650   Median :2.330
##  Mean   :15.762   Mean   :2.401
##  3rd Qu.:17.420   3rd Qu.:2.737
##  Max.   :21.719   Max.   :3.963
```

Uzyskane w ten sposób wyniki możemy określić używając statystyk jakości estymacji lub też wykresów.

```
blad_estymacji_sk <- test$temp - test_sk$var1.pred
summary(blad_estymacji_sk)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.29763 -0.89933 -0.16037 -0.06037  0.94911  3.10812
```

```
MPE <- mean(test$temp - test_sk$var1.pred)
MPE
```

```
## [1] -0.06037294
```

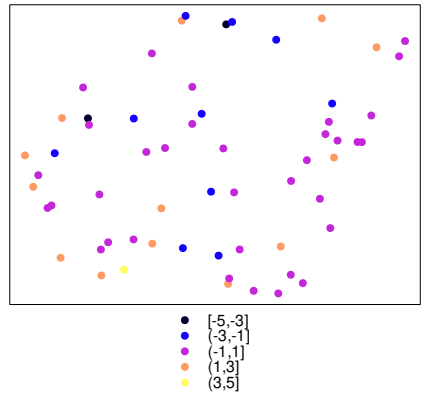
```
RMSE <- sqrt(mean((test$temp - test_sk$var1.pred) ^ 2))
RMSE
```

```
## [1] 1.408253
```

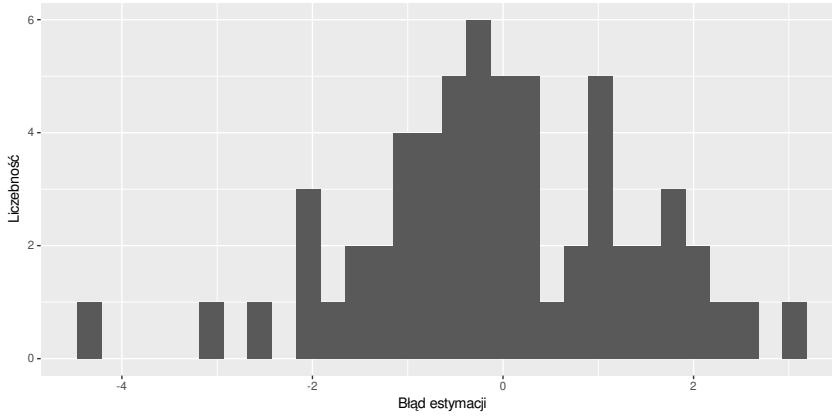
```
R2 <- cor(test$temp, test_sk$var1.pred) ^ 2
R2
```

```
## [1] 0.8504405
```

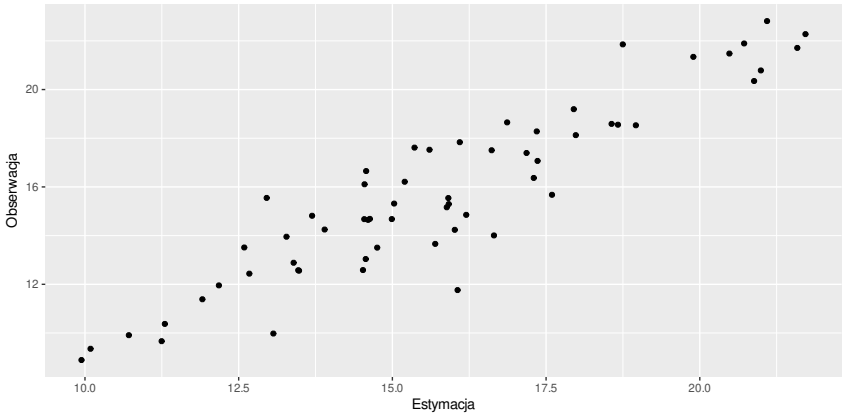
```
test_sk$blad_estymacji_sk <- blad_estymacji_sk
spplot(test_sk, "blad_estymacji_sk", cuts = c(-5, -3, -1, 1, 3, 5))
```




```
ggplot(as.data.frame(test_sk), aes(blad_estymacji_sk)) +
    geom_histogram() +
    xlab("Błąd estymacji") +
    ylab("Liczebność")
```




```
test_sk$true <- test$temp
ggplot(as.data.frame(test_sk), aes(var1.pred, true)) +
    geom_point() +
    xlab("Estymacja") +
    ylab("Obserwacja")
```

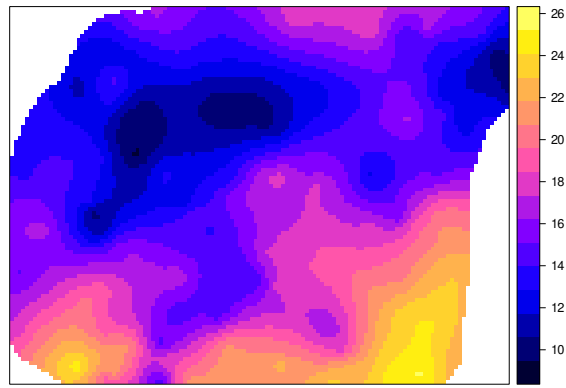



W sytuacji, gdy uzyskany model jest wystarczająco dobry, możemy również uzyskać estymację dla całego obszaru z użyciem funkcji `krige()`, tym razem jednak podając obiekt siatki.

```
test_sk <- krige(temp~1,
                 locations = train,
                 newdata = siatka,
                 model = fitted,
                 beta = 16)
```

```
## [using simple kriging]
```

```
spplot(test_sk, "var1.pred")
```



### 12.3.3. Kroswalidacja

W przypadku kroswalidacji te same dane wykorzystywane są do budowy modelu, estymacji, a następnie do oceny prognozy. Procedura kroswalidacji LOO (ang. *leave-one-out cross-validation*) składa się z poniższych kroków:

1. Zbudowanie matematycznego modelu z dostępnych obserwacji
2. Dla każdej znanej obserwacji następuje:
    - Usunięcie jej ze zbioru danych
    - Użycie modelu do wykonania estymacji w miejscu tej obserwacji
    - Wyliczenie reszty (ang. *residual*), czyli różnicy pomiędzy znaną wartością a estymacją
3. Podsumowanie otrzymanych wyników

W pakiecie **gstat**, kroswalidacja LOO jest dostępna w funkcjach `krige.cv()` oraz `gstat.cv()`. Działają one bardzo podobnie jak funkcje `krige()` oraz `gstat()`, jednak w przeciwieństwie do nich nie wymagają podania obiektu siatki.

```
vario <- variogram(temp~1, data = punkty)
model <- vgm(model = "Sph", nugget = 0.5)
fitted <- fit.variogram(vario, model)
cv_sk <- krige.cv(temp ~ 1,
                  locations = punkty,
                  model = fitted,
                  beta = 16)
summary(cv_sk)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min      max
## x 745546.9 756937.4
## y 712618.8 721192.6
## Is projected: NA
## proj4string : [NA]
## Number of points: 250
## Data attributes:
##    var1.pred         var1.var        observed         residual
##  Min.   : 9.786   Min.   :1.219   Min.   : 8.706   Min.   :-9.007302
##  1st Qu.:13.332   1st Qu.:1.806   1st Qu.:13.284   1st Qu.:-0.953079
##  Median :15.300   Median :2.093   Median :15.309   Median : 0.044585
##  Mean   :15.941   Mean   :2.220   Mean   :15.950   Mean   : 0.008899
##  3rd Qu.:18.062   3rd Qu.:2.477   3rd Qu.:18.273   3rd Qu.: 0.856024
##  Max.   :25.017   Max.   :5.320   Max.   :26.139   Max.   : 6.403263
```

```
##      zscore               fold
##  Min.   :-4.951142   Min.   :  1.00
##  1st Qu.:-0.653052   1st Qu.: 63.25
##  Median : 0.031921   Median :125.50
##  Mean   : 0.005048   Mean   :125.50
##  3rd Qu.: 0.586624   3rd Qu.:187.75
##  Max.   : 3.309199   Max.   :250.00
```

Uzyskane w ten sposób wyniki możemy określić używając statystyk jakości estymacji lub też wykresów.

```
summary(cv_sk$residual)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -9.007302 -0.953079  0.044585  0.008899  0.856024  6.403263
```

```
MPE <- mean(cv_sk$residual)
MPE
```

```
## [1] 0.008898675
```

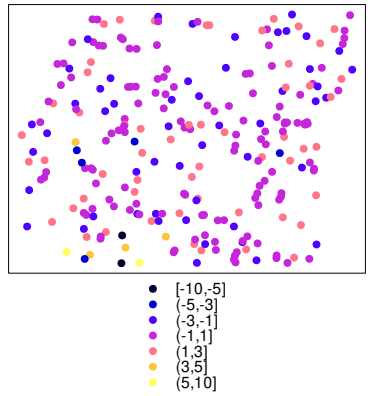
```
RMSE <- sqrt(mean((cv_sk$residual) ^ 2))
RMSE
```

```
## [1] 1.635875
```

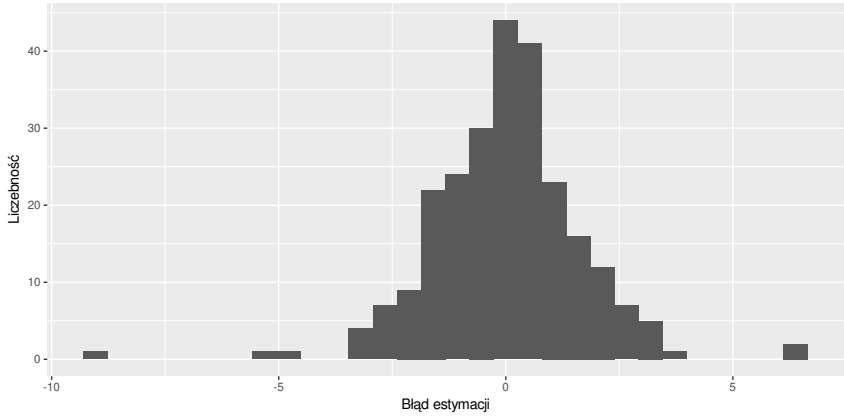
```
R2 <- cor(cv_sk$observed, cv_sk$var1.pred) ^ 2
R2
```

```
## [1] 0.8177634
```

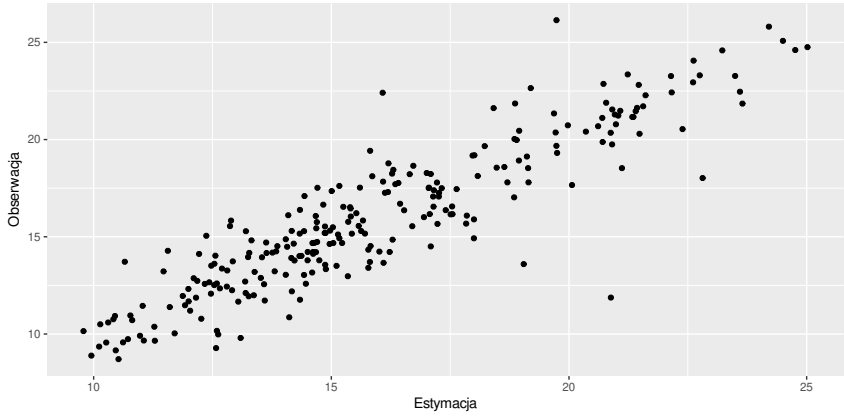
```
spplot(cv_sk, "residual", cuts = c(-10, -5, -3, -1, 1, 3, 5, 10))
```



```
ggplot(as.data.frame(cv_sk), aes(residual)) +
    geom_histogram() +
    xlab("Błąd estymacji") +
    ylab("Liczebność")
```




```
ggplot(as.data.frame(cv_sk), aes(var1.pred, observed)) +
    geom_point() +
    xlab("Estymacja") +
    ylab("Obserwacja")
```

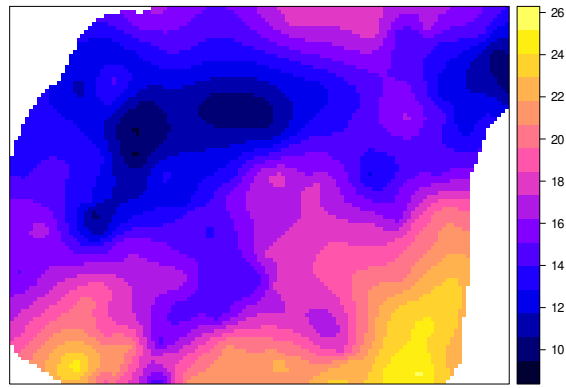



Podobnie jak w walidacji podzbiorem, gdy uzyskany model jest wystarczająco dobry, estymację dla całego obszaru uzyskuje się z użyciem funkcji `krige()`.

```
cv_skk <- krige(temp~1,
                locations = train,
                newdata = siatka,
                model = fitted,
                beta = 16)
```

```
## [using simple kriging]
```

```
spplot(cv_skk, "var1.pred")
```




## 12.4. Zadania

1. Wydziel obiekt `punkty` w taki sposób aby 70% danych należało zbioru treningowy, a 30 % danych do zbioru testowego. Zwizualizuj oba nowe zbiory danych.
2. Stwórz optymalne modele zmiennej `temp` ze zbioru treningowego używając krigingu zwykłego, kokrigingu oraz krigingu uniwersalnego.
3. Wykonaj estymacje korzystając z powyższych modeli. Porównaj uzyskane estymacje korzystając ze statystyk jakości estymacji oraz wizualizacji jakości estymacji i używając zbioru testowego. Który ze stworzonych modeli można uznać za najlepszy? Dlaczego?
4. Porównaj uzyskane modele używając kroswalidacji. Jak wygląda rozkład reszt z uzyskanych estymacji? Który model można uznać za najlepszy oglądając rozkłady reszt?

# 13. Symulacje

Odtworzenie obliczeń z tego rozdziału wymaga załączenia poniższych pakietów oraz wczytania poniższych danych:

```
library(sp)
library(gstat)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
```

## 13.1. Symulacje geostatystyczne

Kriging daje optymalne estymacje, czyli wyznacza najbardziej potencjalnie możliwą wartość dla wybranej lokalizacji. Dodatkowo, efektem krigingu jest wygładzony obraz. W konsekwencji wyniki estymacji różnią się od danych pomiarowych. Uzyskiwana jest tylko (aż?) estymacja, a prawdziwa wartość jest niepewna... Korzystając z symulacji geostatystycznych nie tworzymy estymacji, ale generujemy równie prawdopodobne możliwości poprzez symulację z rozkładu prawdopodobieństwa (wykorzystując generator liczb losowych).

### 13.1.1. Właściwości

Właściwości symulacji geostatystycznych:

- Efekt symulacji ma bardziej realistyczny przestrzenny wzór (ang. *pattern*) niż kriging, którego efektem jest wygładzona reprezentacja rzeczywistości
- Każda z symulowanych map jest równie prawdopodobna
- Symulacje pozwalają na przedstawianie niepewności interpolacji
- Jednocześnie - kriging jest znacznie lepszy, gdy naszym celem jest jak najdokładniejsza estymacja

### 13.1.2. Typy symulacji

Istnieją dwa typy symulacji geostatystycznych:

- Symulacje bezwarunkowe (ang. *Unconditional Simulations*) - wykorzystujące semiwariogram, żeby włączyć informację przestrzenną, ale wartości ze zmierzonych punktów nie są w niej wykorzystywane
- Symulacje warunkowe (ang. *Conditional Simulations*) - opiera się ona o średnią wartość, strukturę kowariancji oraz obserwowane wartości (w tym sekwencyjna symulacja danych kodowanych (ang. *Sequential indicator simulation*))

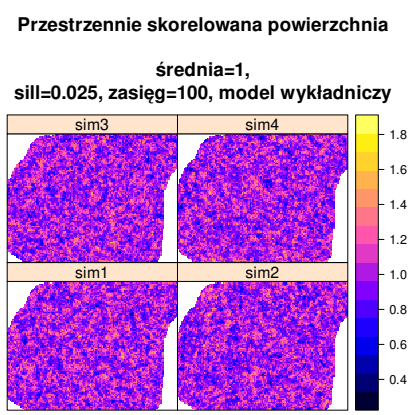
## 13.2. Symulacje bezwarunkowe

Symulacje bezwarunkowe w pakiecie **gstat** tworzy się z wykorzystaniem funkcji `krige()`. Podobnie jak w przypadku estymacji geostatystycznych, należy tutaj podać wzór, model, siatkę, średnią globalną (`beta`), oraz liczbę sąsiednich punktów używanych do symulacji (w przykładzie poniżej `nmax = 30`). Należy wprowadzić również informacje, że nie korzystamy z danych punktowych (`locations = NULL`) oraz że chcemy stworzyć dane sztuczne (`dummy = TRUE`). Ostatni argument (`nsim = 4`) informuje o liczbie symulacji do przeprowadzenia.

```
sym_bezw1 <- krige(formula = z~1,
                   model = vgm(psill = 0.025,
                               model = "Exp",
                               range = 100),
                   newdata = siatka,
                   beta = 1,
                   nmax = 30,
                   locations = NULL,
                   dummy = TRUE,
                   nsim = 4)
```

```
## [using unconditional Gaussian simulation]
```

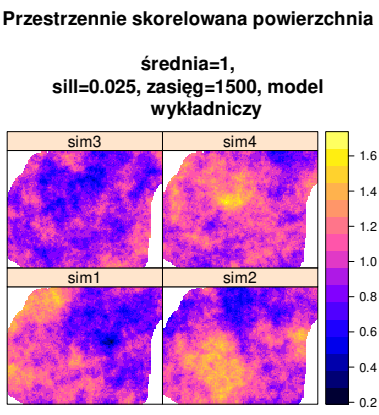
```
spplot(sym_bezw1, main = "Przestrzennie skorelowana powierzchnia
       \nśrednia=1,\nsill=0.025, zasięg=100, model wykładniczy")
```




```
sym_bezw2 <- krige(formula = z~1,
                   model = vgm(psill = 0.025, model = "Exp", range = 1500),
                   newdata = siatka,
                   beta = 1,
                   nmax = 30,
                   locations = NULL,
                   dummy = TRUE,
                   nsim = 4)
```

```
## [using unconditional Gaussian simulation]
```

```
spplot(sym_bezw2, main = "Przestrzennie skorelowana powierzchnia
       \nśrednia=1,\nsill=0.025, zasięg=1500, model
       wykładniczy")
```



## 13.3. Symulacje warunkowe

### 13.3.1. Sekwencyjna symulacja gaussowska

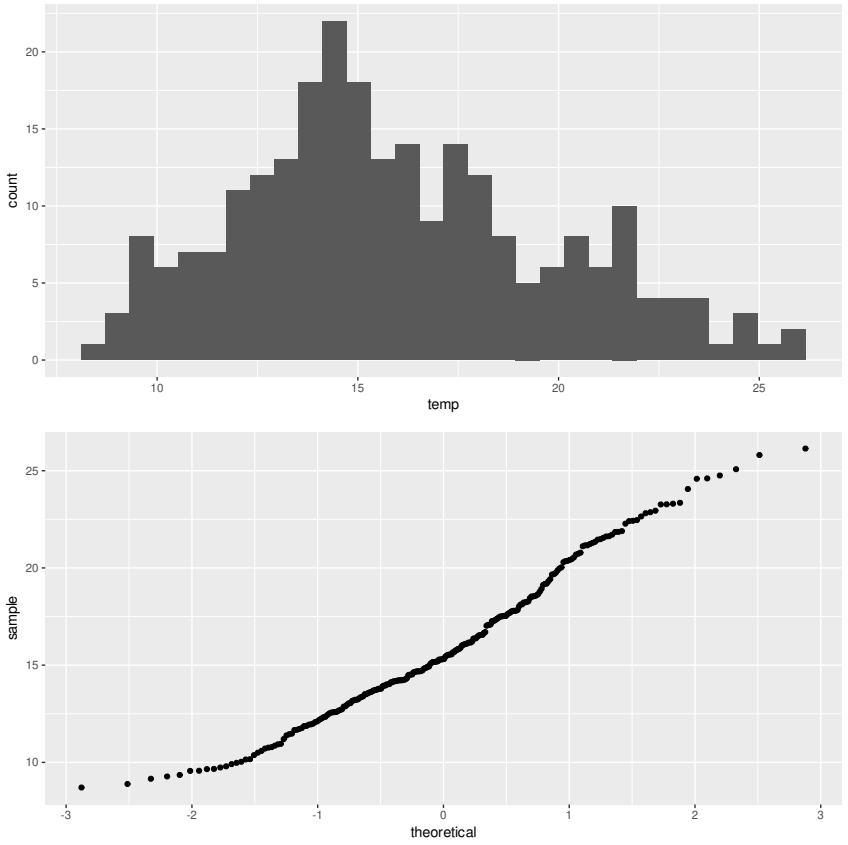
Jednym z podstawowych typów symulacji warunkowych jest sekwencyjna symulacja gaussowska (ang. *Sequential Gaussian simulation*). Polega ona na:

1. Wybraniu lokalizacji nie posiadającej zmierzonej wartości badanej zmiennej
2. Krigingu wartości tej lokalizacji korzystając z dostępnych danych, co pozwala na uzyskanie rozkładu prawdopodobieństwa badanej zmiennej
3. Wylosowaniu wartości z rozkładu prawdopodobieństwa za pomocą generatora liczba losowych i przypisaniu tej wartości do lokalizacji
4. Dodaniu symulowanej wartości do zbioru danych i przejściu do kolejnej lokalizacji
5. Powtórzeniu poprzednich kroków, aż do momentu gdy nie pozostanie już żadna nieokreślona lokalizacja

Sekwencyjna symulacja gaussowska wymaga zmiennej posiadającej wartości o rozkładzie zbliżonym do normalnego. Można to sprawdzić poprzez wizualizacje danych (histogram, wykres kwantyl-kwantyl) lub też test statystyczny (test Shapiro-Wilka). Zmienna `temp` nie ma rozkładu zbliżonego do normalnego.

```
ggplot(punkty@data, aes(temp)) + geom_histogram()
ggplot(punkty@data, aes(sample = temp)) + stat_qq()
shapiro.test(punkty$temp)
```

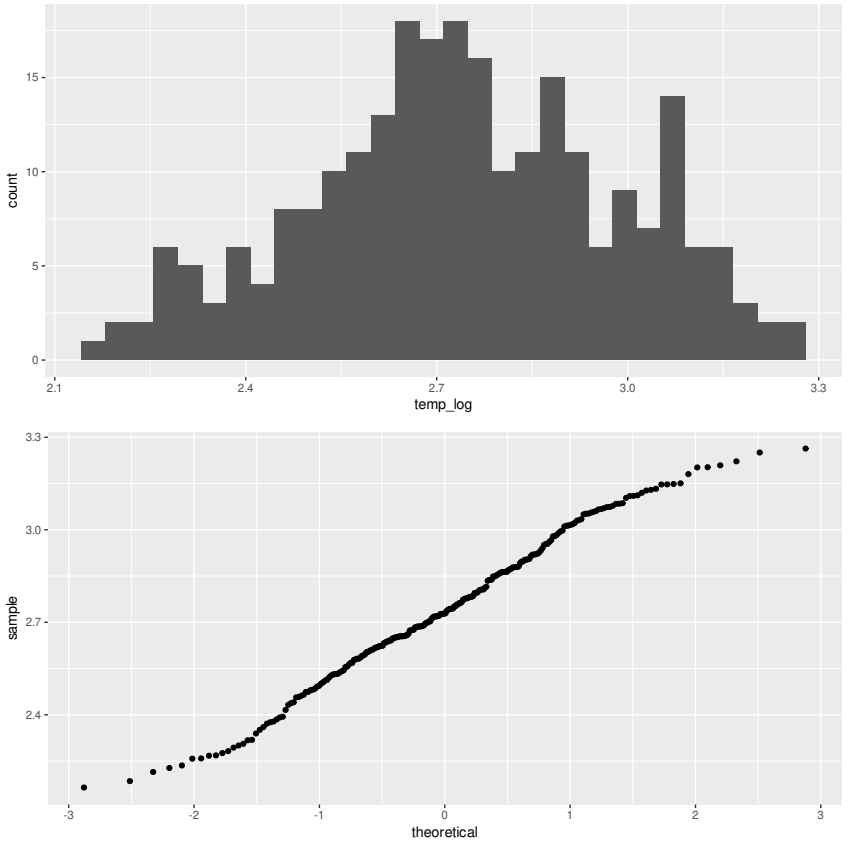
```
##
##  Shapiro-Wilk normality test
##
## data:  punkty$temp
## W = 0.97683, p-value = 0.0004194
```




Na potrzeby symulacji zmienna `temp` została zlogarytmizowna.

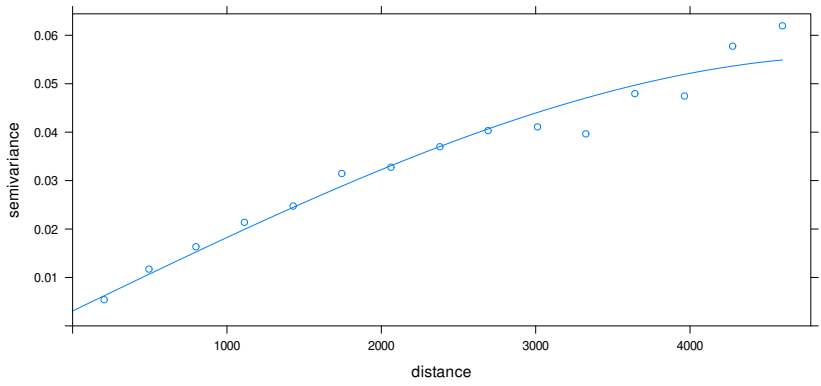
```
punkty$temp_log <- log(punkty$temp)
ggplot(punkty@data, aes(temp_log)) + geom_histogram()
ggplot(punkty@data, aes(sample = temp_log)) + stat_qq()
shapiro.test(punkty$temp_log)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  punkty$temp_log
## W = 0.98907, p-value = 0.05568
```




Dalsze etapy przypominają przeprowadzenie estymacji statystycznej, jedynym wyjątkiem jest dodanie argumentu mówiącego o liczbie symulacji do przeprowadzenia (`nsim` w funkcji `krige()`).

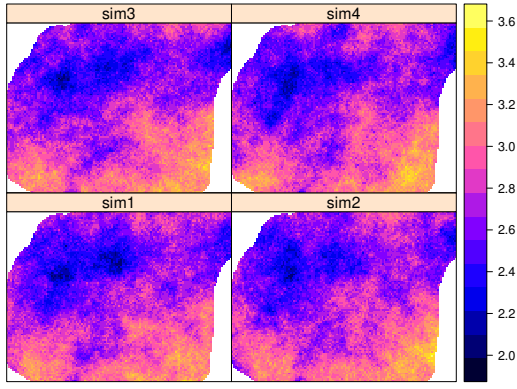
```
vario <- variogram(temp_log~1, locations = punkty)
model <- vgm(model = "Sph", nugget = 0.005)
fitted <- fit.variogram(vario, model)
plot(vario, model = fitted)
```




```
sym_ok <- krige(temp_log~1,
                locations = punkty,
                newdata = siatka,
                model = fitted,
                nmax = 30,
                nsim = 4)
```

```
## drawing 4 GLS realisations of beta...
## [using conditional Gaussian simulation]
```

```
spplot(sym_ok)
```




Wyniki symulacji można przetworzyć do pierwotnej jednostki z użyciem funkcji wykładniczej (`exp`).

```
summary(sym_ok)
```

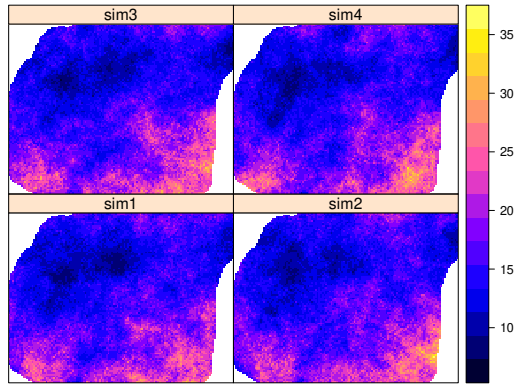
```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##       sim1            sim2            sim3            sim4
##  Min.   :2.001   Min.   :2.025   Min.   :1.985   Min.   :2.060
##  1st Qu.:2.559   1st Qu.:2.579   1st Qu.:2.566   1st Qu.:2.574
##  Median :2.725   Median :2.735   Median :2.719   Median :2.732
##  Mean   :2.730   Mean   :2.746   Mean   :2.741   Mean   :2.745
##  3rd Qu.:2.901   3rd Qu.:2.907   3rd Qu.:2.921   3rd Qu.:2.909
##  Max.   :3.474   Max.   :3.570   Max.   :3.462   Max.   :3.546
```

```
sym_ok@data <- as.data.frame(apply(sym_ok@data, 2, exp))
summary(sym_ok)
```

```
## Object of class SpatialPixelsDataFrame
## Coordinates:
##        min      max
## x 745541.7 756971.7
## y 712616.2 721256.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=19 +k=0.9993 +x_0=500000 +y_0=-5300000
## +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
## Number of points: 10993
## Grid attributes:
##    cellcentre.offset cellsize cells.dim
## s1          745586.7       90       127
## s2          712661.2       90        96
## Data attributes:
##       sim1            sim2            sim3            sim4
```

```
##  Min.   : 7.396   Min.   : 7.578   Min.   : 7.277   Min.   : 7.846
##  1st Qu.:12.928   1st Qu.:13.180   1st Qu.:13.010   1st Qu.:13.113
##  Median :15.250   Median :15.412   Median :15.164   Median :15.367
##  Mean   :15.792   Mean   :16.032   Mean   :15.990   Mean   :16.050
##  3rd Qu.:18.185   3rd Qu.:18.310   3rd Qu.:18.563   3rd Qu.:18.335
##  Max.   :32.278   Max.   :35.532   Max.   :31.871   Max.   :34.692
```

```
spplot(sym_ok)
```




Symulacje geostatystyczne pozwalają również na przedstawianie niepewności interpolacji. W tym wypadku należy wykonać znacznie więcej powtórzeń (np. `nsim = 100`).

```
sym_sk <- krige(temp_log~1,
                location = punkty,
                newdata = siatka,
                model = fitted,
                nsim = 100,
                nmax = 30)
```

```
## drawing 100 GLS realisations of beta...
## [using conditional Gaussian simulation]
```

Uzyskane wyniki należy przeliczyć do oryginalnej jednostki, a następnie wyliczyć odchylenie standardowe ich wartości. Można to zrobić korzystając dwukrotnie z funkcji `apply()`. Przywrócenie oryginalnej jednostki odbywa się poprzez argumenty `MARGIN = 2` oraz `FUN = exp`. Oznacza on, że funkcja `exp()` będzie wykonana na każdej kolumnie. W efekcie tej operacji liczba kolumn nie ulega zmianie.

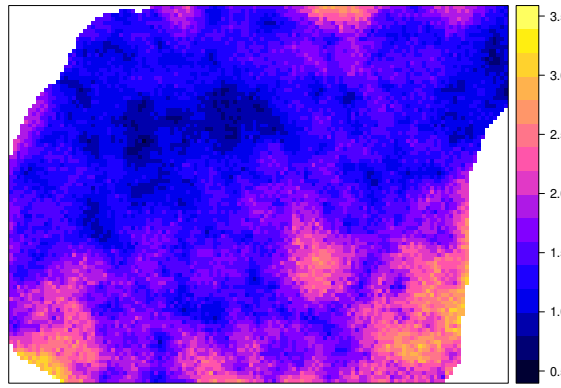
```
# przywrócenie oryginalnej jednostki
sym_sk@data <- data.frame(apply(sym_sk@data,
                                MARGIN = 2,
                                FUN = exp))
```

Wyliczenie odchylenia standardowego odbywa się poprzez argumenty `MARGIN = 1` oraz `FUN = sd`. W ten sposób funkcja `sd()` będzie wykonana na każdym wierszy. W efekcie tej operacji otrzymuje się tylko jedną kolumnę.

```
# przywrócenie wyliczenie odchylenia standardowego
sym_sk@data <- data.frame(apply(sym_sk@data,
                                MARGIN = 1,
                                FUN = sd))
```

Finalnie otrzymujemy mapę odchylenia standardowego symulowanych wartości. Można na niej odczytać obszary o najpewniejszych (najmniej zmiennych) wartościach (niebieski kolor) oraz obszary o największej zmienności cechy (kolor żółty).

```
spplot(sym_sk)
```




## 13.3.2. Sekwencyjna symulacja danych kodowanych

Symulacje geostatystyczne można również zastosować do danych binarnych. Dla potrzeb przykładu tworzona jest nowa zmienna `temp_ind` przyjmująca wartość `TRUE` dla pomiarów o wartościach temperatury niższych niż 12 stopni Celsjusza oraz `FALSE` dla pomiarów o wartościach temperatury równych lub wyższych niż 12 stopni Celsjusza.

```
summary(punkty$temp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   8.706  13.284  15.309  15.950  18.273  26.139
```

```
punkty$temp_ind <- punkty$temp < 12
summary(punkty$temp_ind)
```

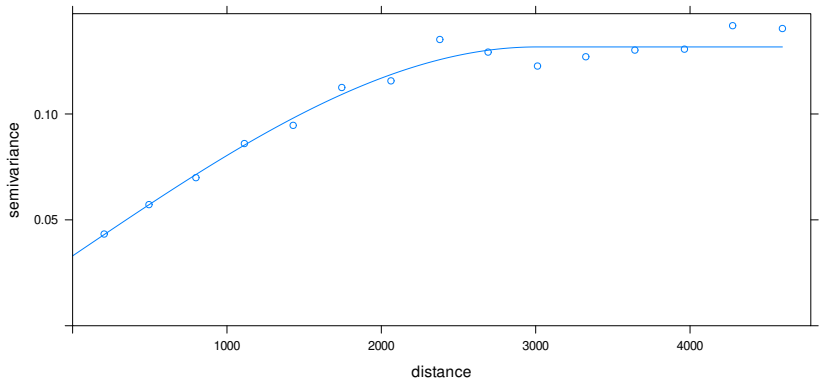
```
##    Mode   FALSE    TRUE
## logical     212      38
```

W tej metodzie kolejne etapy przypominają przeprowadzenie krigingu danych kodowanych. Jedynie w funkcji `krige()` należy dodać argument mówiący o liczbie symulacji do przeprowadzenia (`nsim`).

```
vario_ind <- variogram(temp_ind~1, locations = punkty)
# plot(vario_ind)
fitted_ind <- fit.variogram(vario_ind,
                            vgm(model = "Sph", nugget = 0.3))
fitted_ind
```

```
##   model      psill    range
## 1   Nug 0.03299462    0.000
## 2   Sph 0.09866553 3006.406
```

```
plot(vario_ind, model = fitted_ind)
```

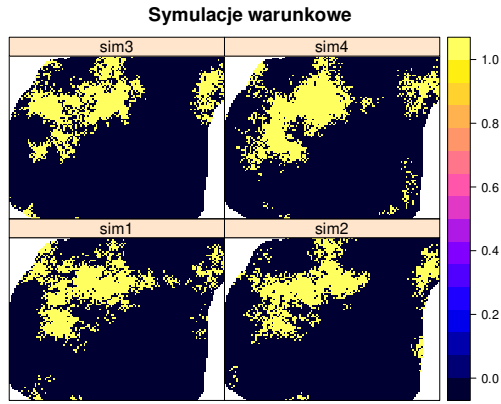


```
sym_ind <- krige(temp_ind~1,
                 locations = punkty,
                 newdata = siatka,
                 model = fitted_ind,
                 indicators = TRUE,
                 nsim = 4,
                 nmax = 30)
```

```
## drawing 4 GLS realisations of beta...
## [using conditional indicator simulation]
```

Wynik symulacji danych kodowanych znacząco różni się od wyniku krigingu danych kodowanych. W przeciwieństwie do tej drugiej metody, w rezultacie symulacji nie otrzymujemy prawdopodobieństwa zajścia danej klasy, ale konkretne wartości `1` lub `0`.

```
spplot(sym_ind, main = "Symulacje warunkowe")
```




## 13.4. Zadania

1. Stwórz nową siatkę dla obszaru o zasięgu x od 0 do 40000 i zasięgu y od 0 do 30000 oraz rozdzielczości 250.
2. Zbuduj po trzy symulacje bezwarunkowe w tej siatce korzystając z:
    - modelu sferycznego o semiwariancji cząstkowej 15 i zasięgu 7000
    - modelu nugetowego o wartości 1 razem z modelem sferycznym o semiwariancji cząstkowej 15 i zasięgu 7000

- modelu sferycznego o semiwariancji cząstkowej 5 i zasięgu 7000
- modelu sferycznego o semiwariancji cząstkowej 15 i zasięgu 1000

3. Porównaj graficznie uzyskane powyżej wyniki i opisz je.
4. Stwórz optymalny model semiwariogramu zmiennej `temp` z obiektu `punkty`. Następnie korzystając z wybranej metody krigingowej, poznanej w poprzednich rozdziałach, wykonaj estymację zmiennej `temp`.
5. Wykonaj cztery symulacje warunkowe używając optymalnego modelu semiwariancji stworzonego w poprzednim zadaniu. Porównaj uzyskaną estymację geostatystyczną z symulacjami geostatystycznymi. Jakie można zaobserwować podobieństwa a jakie różnice?
6. Zbuduj optymalny model semiwariogramu empirycznego określający prawdopodobieństwo wystąpienia wartości `ndvi` poniżej 0.1 dla zbioru `punkty`. Wykonaj estymację metodą krigingu danych kodowanych. Następnie używając tego samego modelu, wykonaj cztery symulacje warunkowe (symulacje danych kodowanych). Jakie wartości progowe prawdobodobieństwa przypominają uzyskane symulacje?

# A. Źródła wiedzy

## A.1. Podstawy R

- An Introduction to R[1] - oficjalne wprowadzenie do R
- Przewodnik po pakiecie R[2]
- Programowanie w języku R[3]
- Hands-On Programming with R[4]
- R for Data Science[5] i jej polskie tłumaczenie Język R. Kompletny zestaw narzędzi dla analityków danych[6]

## A.2. Analizy przestrzenne w R

- Geocomputation with R[7]
- Applied Spatial Dala Analysis with R[8]
- CRAN Task View: Analysis of Spatial Data[9]

## A.3. Geostatystyka

- Praktyczny poradnik - jak szybko zrozumieć i wykorzystać geostatystykę w pracy badawczej[10]
- A Practical Guide to Geostatistical Mapping[11]
- gstat user's manual[12]

[1] http://cran.r-project.org/doc/manuals/R-intro.pdf
[2] http://www.biecek.pl/R/
[3] http://rksiazka.rexamine.com/
[4] https://rstudio-education.github.io/hopr/
[5] http://r4ds.had.co.nz/
[6] https://helion.pl/ksiazki/jezyk-r-kompletny-zestaw-narzedzi-dla-analitykow-danych-hadley-wickham-garrett-grolemund,jezrko.htm#format/b
[7] http://robinlovelace.net/geocompr/
[8] http://www.asdar-book.org/
[9] https://cran.r-project.org/web/views/Spatial.html
[10] http://www.geoinfo.amu.edu.pl/staff/astach/www_geostat/programy/A_Stach_%20poradnik_geostatystyki.pdf
[11] http://spatial-analyst.net/book/system/files/Hengl_2009_GEOSTATe2c1w.pdf
[12] http://www.gstat.org/gstat.pdf

- Applied Geostatistics[13]
- Statistics for spatial data[14]
- Geostatistics for Natural Resources Evaluation[15]
- Geostatistics for Environmental Scientists[16]

[13]https://books.google.pl/books?id=vC2dcXFLI3YC
[14]https://books.google.pl/books?id=4SdRAAAAMAAJ
[15]http://www.oupcanada.com/catalog/9780195115383.html
[16]http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470028580.html

# B. Dane

Dane wykorzystywane w tym skrypcie można pobrać w postaci spakowanego archiwum (dla rozdziału 2) oraz korzystając z pakietu **geostatbook** (dla kolejnych rozdziałów). Dodatkowo, przy instalacji pakietu **geostatbook** pobierane są wszystkie inne pakiety potrzebne do pełnego korzystania z materiałów zawartych w tym skrypcie.

- Archiwum zawierające dane do rozdziału drugiego[1]
- Dane do kolejnych rozdziałów są zawarte w pakiecie geostatbook:[2]

```
# install.packages("devtools")
devtools::install_github("nowosad/geostatbook@2")
```

## B.1. Zbiory danych w pakiecie geostatbook

```
library(geostatbook)
```

### B.1.1. `punkty`

```
data("punkty")
?punkty
```

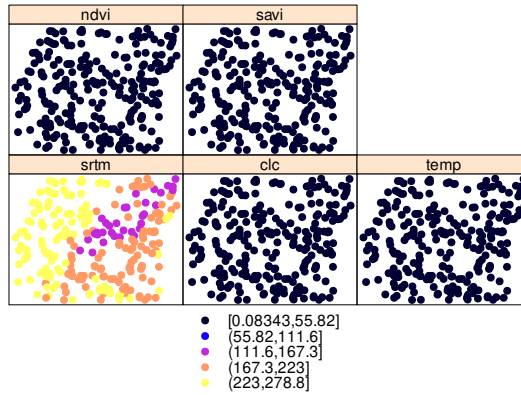
Zbiór danych `punkty` zawiera 252 obserwacje oraz 5 zmiennych dla obszaru Suwalskiego Parku Krajobrazowego i okolic. Zmienne:

- srtm - wysokość w metrach n.p.m. pozyskana z numerycznego modelu terenu z misji SRTM
- clc - uproszczona kategoria pokrycia terenu z bazy Corine Land Cover. 1 oznacza tereny rolne, 2 oznacza lasy i ekosystemy seminaturalne, 3 to obszary podmokłe, 4 to obszary wodne
- temp - temperatura powierzchni ziemi w stopniach Celsjusza

[1] https://github.com/Nowosad/geostat_book/blob/master/dane-wyd2.zip?raw=true
[2] https://github.com/Nowosad/geostatbook

- ndvi - znormalizowany różnicowy wskaźnik wegetacji (ang. *Normalized Difference Vegetation Index*)
- savi - ang. *Soil Adjusted Vegetation Index*

```
spplot(punkty)
```



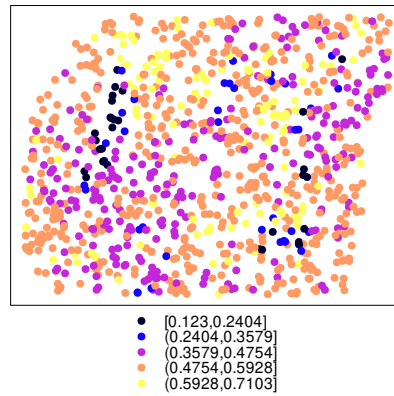

### B.1.2. `punkty_ndvi`

```
data("punkty_ndvi")
?punkty_ndvi
```

Zbiór danych `punkty_ndvi` zawiera 1011 obserwacji oraz 1 zmienną dla obszaru Suwalskiego Parku Krajobrazowego i okolic. Zmienna:

- ndvi - znormalizowany różnicowy wskaźnik wegetacji (ang. *Normalized Difference Vegetation Index*)

```
spplot(punkty_ndvi)
```



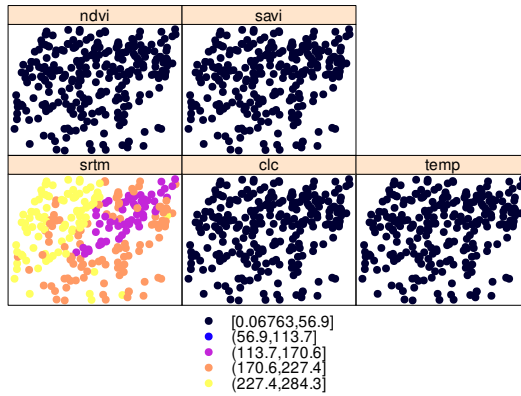

### B.1.3. `punkty_pref`

```
data("punkty_pref")
?punkty_pref
```

Zbiór danych `punkty_pref` zawiera 239 obserwacji oraz 1 zmiennych dla obszaru Suwalskiego Parku Krajobrazowego i okolic. Są one rozlokowane w sposób preferencyjny. Zmienna:

- srtm - wysokość w metrach n.p.m. pozyskana z numerycznego modelu terenu z misji SRTM
- clc - uproszczona kategoria pokrycia terenu z bazy Corine Land Cover. 1 oznacza tereny rolne, 2 oznacza lasy i ekosystemy seminaturalne, 3 to obszary podmokłe, 4 to obszary wodne
- temp - temperatura powierzchni ziemi w stopniach Celsjusza
- ndvi - znormalizowany różnicowy wskaźnik wegetacji (ang. *Normalized Difference Vegetation Index*)
- savi - ang. *Soil Adjusted Vegetation Index*

```
spplot(punkty_pref)
```

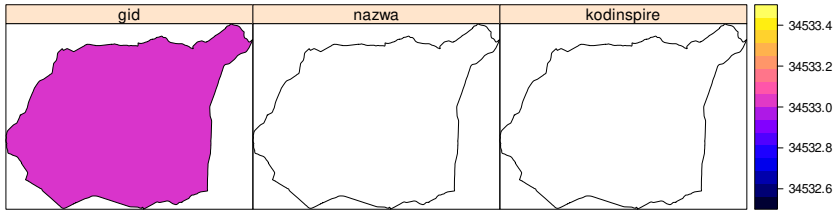



### B.1.4. granica

```
data("granica")
?granica
```

Granica Suwalskiego Parku Krajobrazowego.

```
spplot(granica)
```



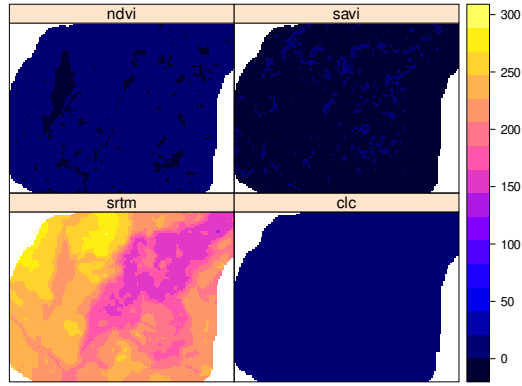

### B.1.5. siatka

```
data("siatka")
?siatka
```

Siatka badanego obszaru dla obszaru Suwalskiego Parku Krajobrazowego i okolic. Zawiera ona ona 127 wierszy i 96 kolumn oraz 4 zmienne:

- srtm - wysokość w metrach n.p.m. pozyskana z numerycznego modelu terenu z misji SRTM
- clc - uproszczona kategoria pokrycia terenu z bazy Corine Land Cover. 1 oznacza tereny rolne, 2 oznacza lasy i ekosystemy seminaturalne, 3 to obszary podmokłe, 4 to obszary wodne
- ndvi - znormalizowany różnicowy wskaźnik wegetacji (ang. *Normalized Difference Vegetation Index*)
- savi - ang. *Soil Adjusted Vegetation Index*

```
spplot(siatka)
```



# C. Powtarzalne przykłady

## C.1. Co to?

Powtarzalny przykład oznacza fragment kodu, który może być odtworzony przez inną osobę na innym komputerze/przez siebie samego w przyszłości.

Powtarzalny przykład może służyć pokazaniu poprawnego rozwiązania, wskazaniu na błędy w funkcjach, lub też jako załącznik do prośby o pomoc z kodem.

Powtarzalny przykład powinien składać się przynajmniej z:

- Z małego zbioru danych wystarczającego do odtworzenia obliczeń
- Krótkiego kodu, który może być uruchomiony na powyższym zbiorze danych
- Czasem ważne są dodatkowe informacje o używanej wersji R, posiadanym systemie operacyjnym, wersjach używanych pakietów, etc. Można do tego użyć funkcji `sessionInfo()`

Źródło: stackoverflow.com[1]

## C.2. Pakiet reprex

Stworzenie powtarzalnego przykładu w R może zostać ułatwione poprzez stosowanie pakietu **reprex**. Ten pakiet uruchamia wybraną część kodu, wykonuje kolejne operacje, a następnie zapisuje uzyskany wynik do schowka.

Pakiet **reprex** można zainstalować poprzez funkcję `install.packages()`:

```
install.packages("reprex")
```

Główną funkcją w tym pakiecie jest `reprex()`. Funkcję `reprex()` można też użyć poprzez wpisanie kodu wewnątrz tej funkcji lub też poprzez wybranie opcji `Reprex selection` z meny Addins w RStudio. Możliwe jest również stworzenie powtarzalnego przykładu na podstawie skryptu R:

[1] https://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example

```
reprex(input = "moj_skrypt.R")
```

## C.3. Tworzenie powtarzalnego przykładu

### C.3.1. Prosty przykład

Sprawdźmy działanie pakietu **reprex** na prostym przykładzie - tworzymy dwa obiekty `x` i `y`, nadajemy im wartości a następnie mnożymy je przez siebie:

```
library(reprex)
reprex({
        x <- 1
        y <- 5
        x * y
})
```

Po jego uruchomieniu otrzymujemy wynik zapisany w schowku jako Markdown oraz w postaci wyświetlonego HTMLa:

```
x <- 1
y <- 5
x * y
#> [1] 5
```

Created on 2018-09-14 by the reprex package (v0.2.0).

### C.3.2. Złożony przykład

Spróbujmy teraz trochę bardziej skomplikowanego przykładu. Naszym celem jest stworzenie mapy punktowej temperatury na podstawie obiektu **punkty**. Powtarzalny przykład może posłużyć do szybkiego określenia problemów z kodem:

```
library(reprex)
reprex({
        library(sp)
        data(punkty)
        spplot(punkty, "temperatura")
})
```

Powyższy kod ma dwa problemy - czy jesteś w stanie je wskazać?

```
library(sp)
data(punkty)
#> Warning in data(punkty): data set 'punkty' not found
spplot(punkty, "temperatura")
#> Error in spplot(punkty, "temperatura"): object 'punkty' not found
```

Created on 2018-09-14 by the reprex package (v0.2.0).

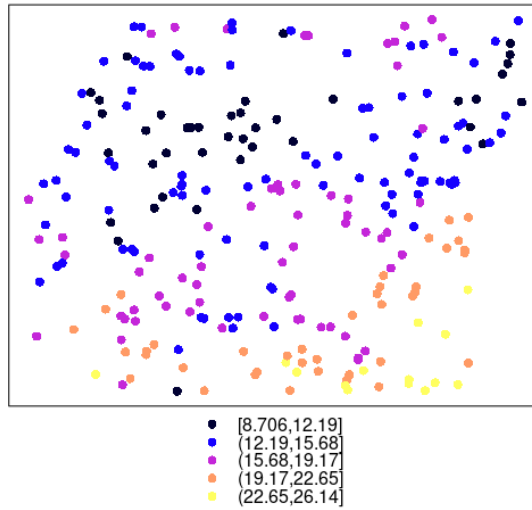
Odpowiedź - ten kod nie jest w pełni samowystarczalny - brakuje tam dołączenia pakietu **geostatbook**, który zawiera zbiór danych `punkty`. Drugi problem to użyta zmienna do wizualizacji - `"temperatura"` nie istnieje w zbiorze danych `punkty`. Zamiast niej powinna być użyta poprawna nazwa zmiennej - `temp`. Naprawiona wersja tego kodu znajduje się poniżej:

```
library(sp)
data(punkty)
#> Warning in data(punkty): data set 'punkty' not found
spplot(punkty, "temperatura")
#> Error in spplot(punkty, "temperatura"): object 'punkty' not found
```

Created on 2018-09-14 by the reprex package (v0.2.0).

```
library(reprex)
reprex({
        library(sp)
        library(geostatbook)
        data(punkty)
        spplot(punkty, "temp")
})
```

```
library(sp)
library(geostatbook)
data(punkty)
spplot(punkty, "temp")
```




Created on 2018-09-14 by the reprex package (v0.2.0).

W efekcie otrzymujemy nie tylko kod użyty do obliczeń, ale również wynikową grafikę.

## C.4. Więcej informacji

- Oficjalna strona pakietu reprex[2]
- So you've been asked to make a reprex[3]
- How to make a great R reproducible example[4]
- Magic reprex[5]
- reprex: help me help you![6]
- Get help![7]

[2]https://reprex.tidyverse.org/index.html
[3]https://www.jessemaegan.com/post/so-you-ve-been-asked-to-make-a-reprex/
[4]https://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example
[5]https://www.njtierney.com/post/2017/01/11/magic-reprex/
[6]https://speakerdeck.com/jennybc/reprex-help-me-help-you
[7]https://www.tidyverse.org/help/#reprex

# D. Przykład analizy geostatystycznej

## D.1. Analiza geostatystyczna

Analiza geostatystyczna jest złożonym procesem, często wymagającym sprawdzenia jakości danych i ich korekcji oraz wypróbowania wielu możliwości modelowania. Poniższy suplement skupia się na pokazaniu przykładu uproszczonej analizy geostatystycznej, w której głównym celem jest estymacja średniej wartości temperatury.

## D.2. Przygotowanie danych

Pierwszym krokiem analizy geostatystycznej jest załadowanie pakietów, które zostaną użyte. Oczywiście, brakujące pakiety można załadować także w trakcie analizy geostatystycznej.

```
library(sp)
library(rgdal)
library(gstat)
library(ggplot2)
```

Kolejnym krokiem jest wczytanie danych oraz sprawdzenie ich jakości.

```
moje_punkty <- readOGR("dane/moje_punkty.gpkg")
```

```
## OGR data source with driver: GPKG
## Source: "/home/jn/Tmp/geostat_book-2/dane/moje_punkty.gpkg", layer: "moje_punkty"
## with 190 features
## It has 1 fields
```

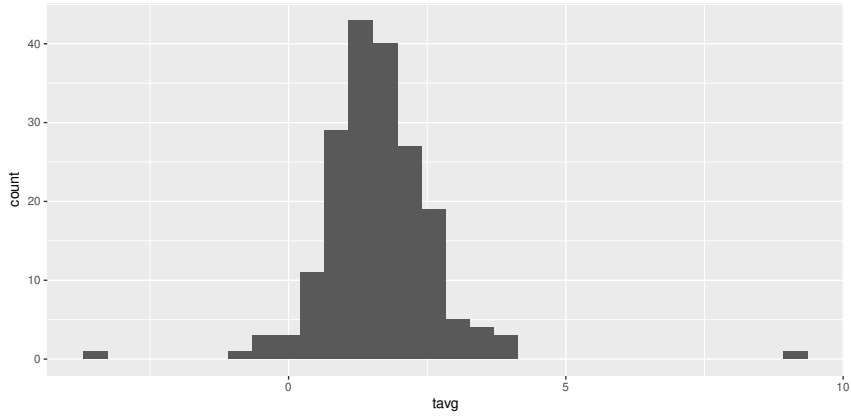
```
summary(moje_punkty)
```

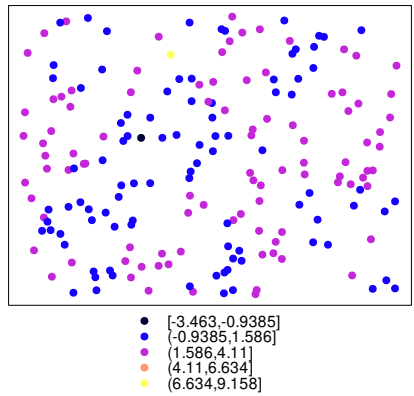
```
## Object of class SpatialPointsDataFrame
## Coordinates:
```

```
##                min      max
## coords.x1 345007.1 357010.0
## coords.x2 312609.0 321540.5
## Is projected: NA
## proj4string : [NA]
## Number of points: 190
## Data attributes:
##       tavg
##  Min.   :-3.463
##  1st Qu.: 1.081
##  Median : 1.647
##  Mean   : 1.633
##  3rd Qu.: 2.181
##  Max.   : 9.158
```

Obiekt `moje_punkty` zawiera tylko jedną zmienną `tavg`, która ma być użyta do stworzenia estymacji. Warto zwizualizować rozkład wartości tej zmiennej w postaci histogramu oraz mapy:

```
ggplot(moje_punkty@data, aes(tavg)) + geom_histogram()
spplot(moje_punkty)
```






Można w ten sposób zauważyć, że występują co najmniej dwie wartości odstające.

```
moje_punkty@data[moje_punkty$tavg == max(moje_punkty$tavg), ]
```

```
## [1] 9.158156
```

```
moje_punkty@data[moje_punkty$tavg == min(moje_punkty$tavg), ]
```

```
## [1] -3.462651
```

Jedna z nich ma wartość ok. -3.5 °C i jest znacznie niższa od pozostałych, druga natomiast jest znacznie wyższa od pozostałych i ma wartość ok. 9.2 °C. Należy w tym momencie zastanowić się czy te wartości odstające są prawidłowymi wartościami, czy też są one błędne. W tej sytuacji, nie posiadając zewnętrznej informacji, bezpieczniej jest usunąć te dwa pomiary. Można to zrobić wyszukując id punktów za pomocą pakietu **mapview**.

Teraz id punktów można użyć do ich wybrania i zastąpienia potencjalnie błędnych wartości wartościami `NA`.

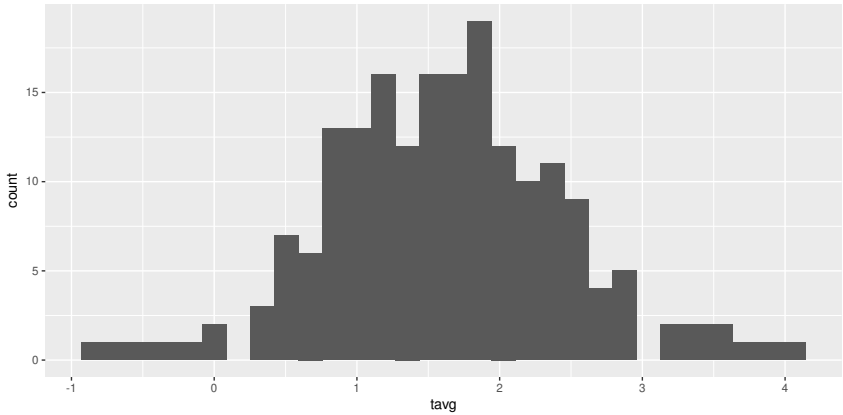
```
# usunięcie wartości według id
moje_punkty@data[60, "tavg"] = NA
moje_punkty@data[72, "tavg"] = NA
```

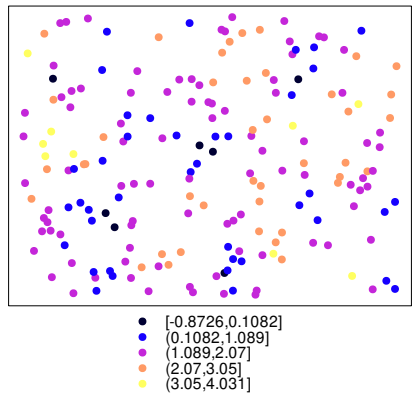
Te punkty nadal istnieją jednak w obiekcie `moje_punkty`. Można je usunąć korzystając z funkcji `is.na` oraz indeksowania:

```
moje_punkty = moje_punkty[!is.na(moje_punkty$tavg), ]
```

Po tej zmianie powinno się po raz kolejny obejrzeć dokładnie dane w celu stwierdzenia, czy problem został naprawiony i czy nie występują dodatkowe sytuacje problemowe.

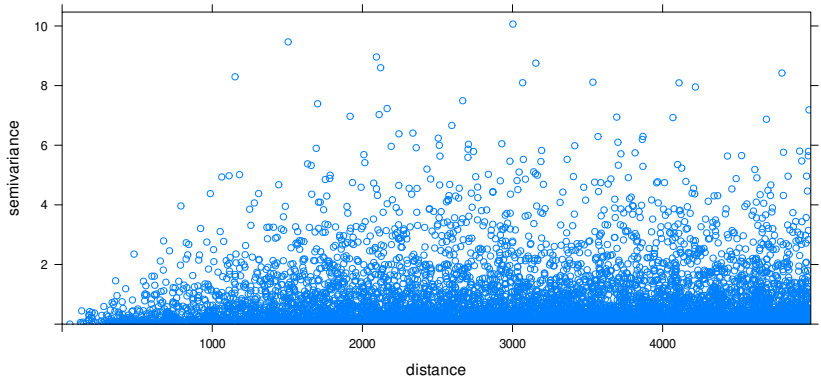
```
ggplot(moje_punkty@data, aes(tavg)) + geom_histogram()
spplot(moje_punkty)
```




Można dodatkowo stworzyć chmurę semiwariogramu w celu wyszukania potencjalnych wartości lokalnie odstających.

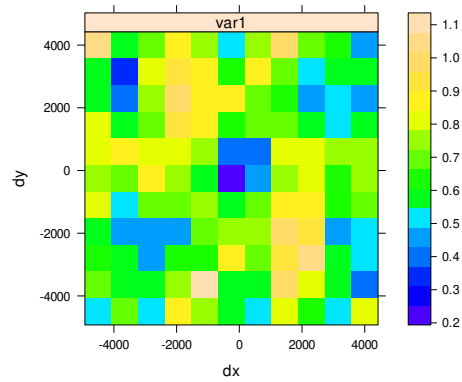
```
moja_chmura = variogram(tavg~1, moje_punkty, cloud = TRUE)
plot(moja_chmura)
```




## D.3. Tworzenie modeli semiwariogramów

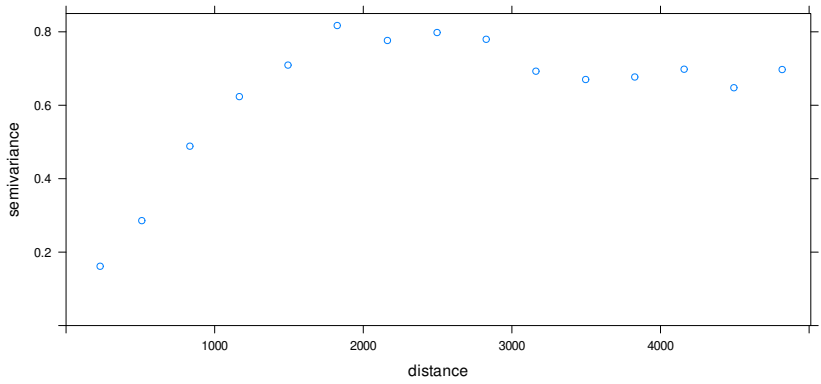
Posiadając już poprawne dane można sprawdzić czy badane zjawisko wykazuje anizotropię przestrzenną poprzez stworzenie mapy semiwariogramu.

```
moja_mapa <- variogram(tavg~1,
                       locations = moje_punkty,
                       cutoff = 4500,
                       width = 850,
                       map = TRUE)
plot(moja_mapa, threshold = 30, col.regions = topo.colors(n = 40))
```

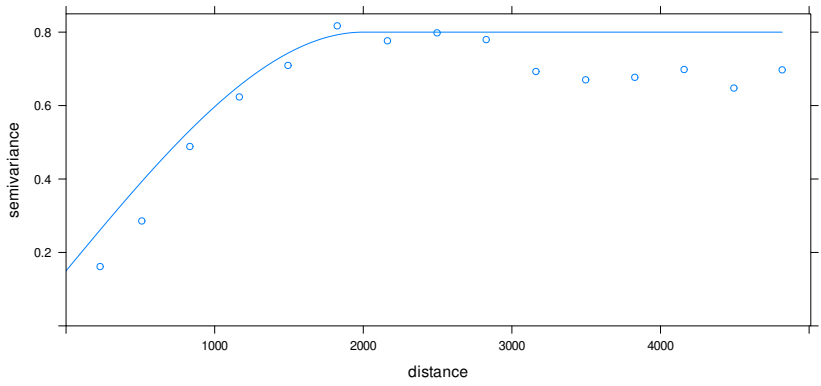


Uzyskana mapa nie pozwala na jednoznaczne stwierdzenie kierunkowej zmienności podobieństwa badanej cechy, w związku z tym można skupić się na modelowaniu izotropowym. Kolejnym etapem jest stworzenie semiwariogramu oraz jego modelowanie. Optymalnie tworzonych jest więcej niż model semiwariogramu, co pozwala na porównanie uzyskanych wyników i wybór lepszego modelu. Do tego przykładu zostały stworzone dwa modele semiwariogramu. Pierwszy z nich używa tylko zmiennej `tavg` oraz modelu ręcznego o wybranych parametrach.

```
moj_semiwar = variogram(tavg~1,
                        locations = moje_punkty)
plot(moj_semiwar)
```

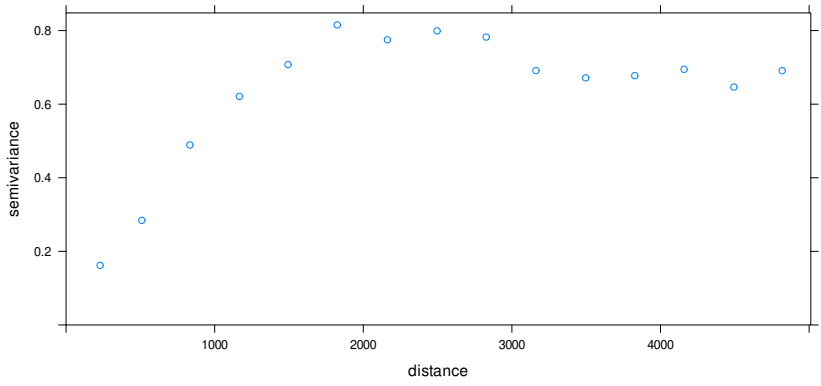



```
moj_model <- vgm(psill = 0.65,
        model = "Sph",
        range = 2000,
        nugget = 0.15)
plot(moj_semiwar, moj_model)
```




Drugi model, oprócz zmiennej `tavg`, używa też wartości współrzędnych oraz modelu o parametrach zmodyfikowanych przez funkcję `fit.variogram()`.

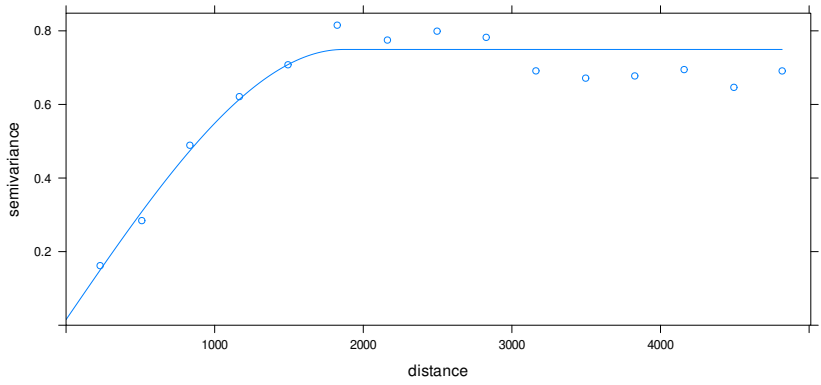
```
moj_semiwar2 <- variogram(tavg~coords.x1 + coords.x2,
                          locations = moje_punkty)
plot(moj_semiwar2)
```




```
moj_model2 <- vgm(model = "Sph", nugget = 0.1)
moj_model2 <- fit.variogram(moj_semiwar2, moj_model2)
moj_model2
```

```
##   model      psill    range
## 1   Nug 0.01521914    0.000
## 2   Sph 0.73418974 1866.631
```

```
plot(moj_semiwar2, moj_model2)
```




## D.4. Ocena jakości semiwariancji

Aby porównać oba modele należy przyjąć metodę walidacji oraz współczynnik jakości estymacji. W tym przykładzie użyto kroswalidacji metodą LOO (funkcja `krige.cv`) oraz pierwiastek średniego błędu kwadratowego (RMSE) jako miary jakości.

```
ocena1 <- krige.cv(tavg~1,
                   locations = moje_punkty,
                   model = moj_model,
                   beta = 30)
RMSE1 <- sqrt(mean((ocena1$residual) ^ 2))
```

```
ocena2 <- krige.cv(tavg~coords.x1 + coords.x2,
                   locations = moje_punkty,
                   model = moj_model2)
RMSE2 <- sqrt(mean((ocena2$residual) ^ 2))
```

Porównanie dwóch wartości RMSE pozwala zdecydowanie stwierdzić, że drugi model charakteryzuje się lepszą jakością estymacji.

```
RMSE1
```

```
## [1] 6.193243
```

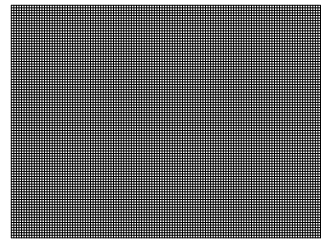
```
RMSE2
```

```
## [1] 0.587956
```

## D.5. Stworzenie siatki

Przedostatnim krokiem jest utworzenie siatki do estymacji.

```
moja_siatka = expand.grid(
  coords.x1 = seq(from = 345050, to = 357050, by = 100),
  coords.x2 = seq(from = 312650, to = 321650, by = 100)
)
coordinates(moja_siatka) <- ~coords.x1 + coords.x2
gridded(moja_siatka) <- TRUE
plot(moja_siatka)
```



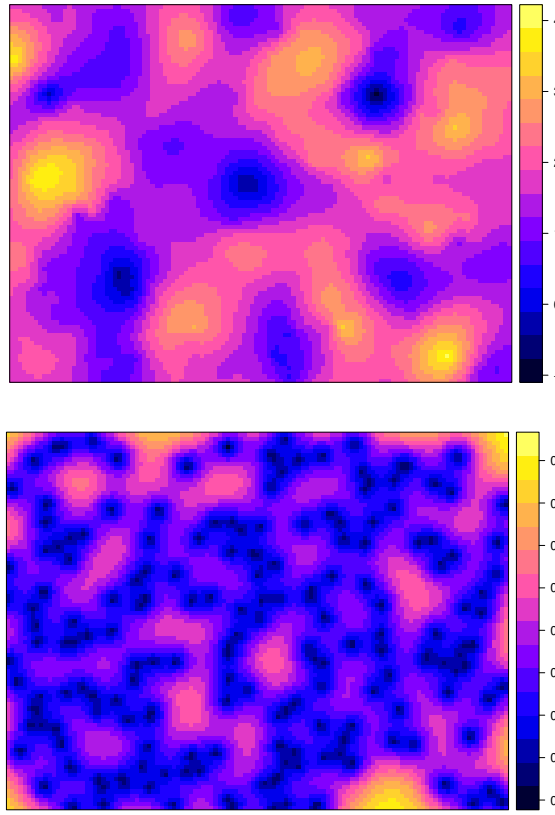
## D.6. Stworzenie estymacji

Następnie nowo utworzona siatka może posłużyć do stworzenia estymacji.

```
moja_estymacja <- krige(tavg~coords.x1 + coords.x2,
              locations = moje_punkty,
              newdata = moja_siatka,
              model = moj_model2)
```

```
## [using universal kriging]
```

```
spplot(moja_estymacja[1])
spplot(moja_estymacja[2])
```



# Bibliografia


Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2019). *rmarkdown: Dynamic Documents for R*. R package version 2.0.

Appelhans, T., Detsch, F., Reudenbach, C., and Woellauer, S. (2019). *mapview: Interactive Viewing of Spatial Data in R*. R package version 2.7.0.

Auguie, B. (2017). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.3.

Bivand, R., Keitt, T., and Rowlingson, B. (2019). *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. R package version 1.4-8.

Bivand, R. and Rundel, C. (2019). *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*. R package version 0.5-2.

Giraudoux, P. (2018). *pgirmess: Spatial Analysis and Data Mining for Field Ecologists*. R package version 1.6.9.

Hijmans, R. J., Phillips, S., Leathwick, J., and Elith, J. (2017). *dismo: Species Distribution Modeling*. R package version 1.1-4.

Kuhn, M. (2018). *caret: Classification and Regression Training*. R package version 6.0-81.

Nowosad, J. (2020). *geostatbook: Geostatystyka w R*. R package version 0.2.9.

Nychka, D., Furrer, R., Paige, J., Sain, S., Gerber, F., and Iverson, M. (2019). *fields: Tools for Spatial Data*. R package version 10.0.

Pebesma, E. and Bivand, R. (2019). *sp: Classes and Methods for Spatial Data*. R package version 1.3-2.

Pebesma, E. and Graeler, B. (2020). *gstat: Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation*. R package version 2.0-4.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., and Yutani, H. (2019). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.2.1.

Xie, Y. (2020a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.17.

Xie, Y. (2020b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.27.