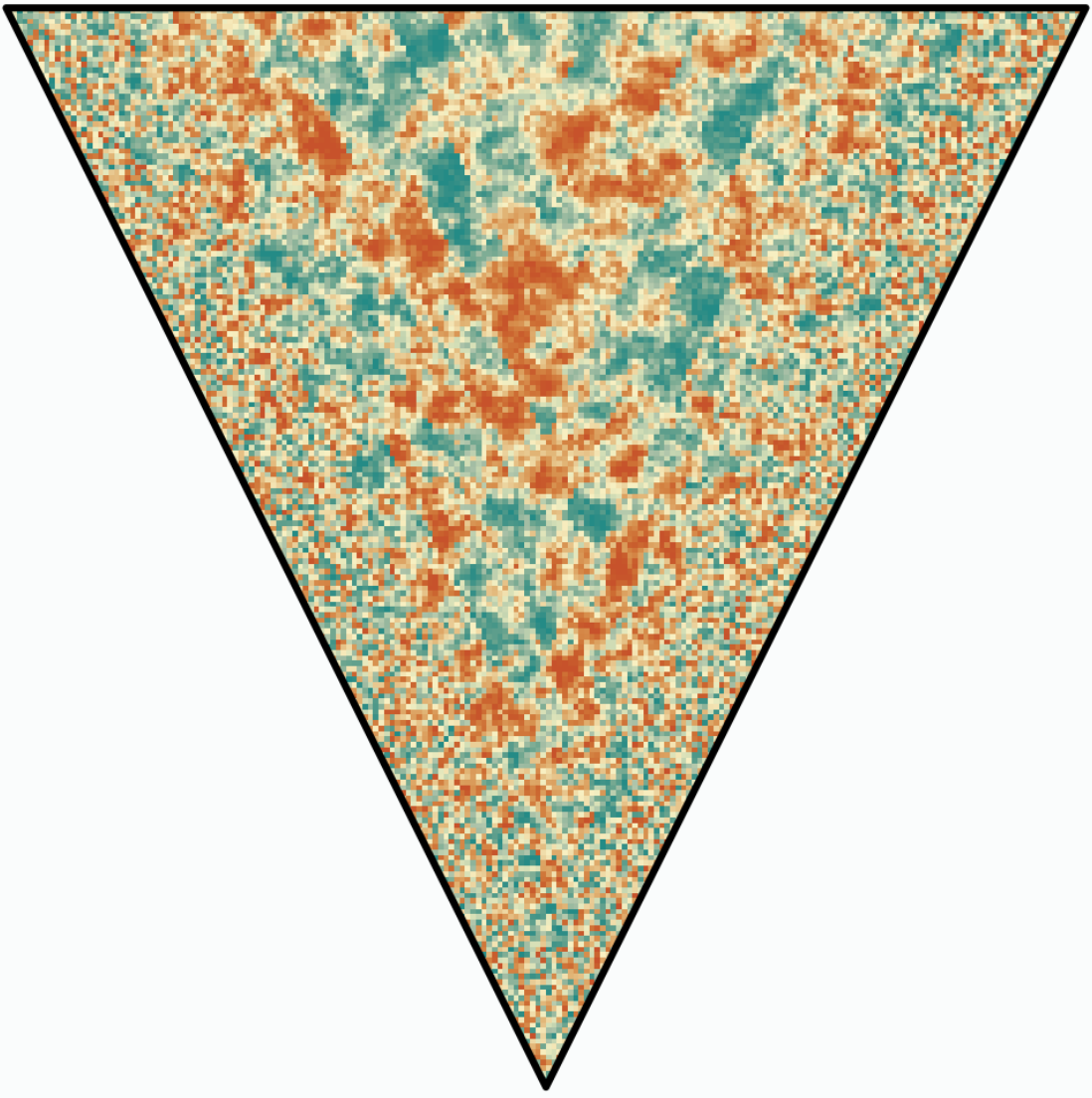
# Geostatystyka w R



Jakub Nowosad

# Geostatystyka w R

Jakub Nowosad

Wydanie trzecie, Poznań 2021
Space A

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

ISBN: 978-83-953296-2-3

# 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 trzy appendiksy. W appendiksie A można znaleźć odnośniki do innych materiałów związanych z geostatystyką i R, appendiks B opisuje dane użyte w tym skrypcie, a appendiks C pokazuje uproszczony przykład analizy geostatystycznej.

Jest to trzecie wydanie tego skryptu. Wersje PDF wydania pierwszego i drugiego można znaleźć pod adresem https://nowosad.github.io/publications/. Główne zmiany względem drugiego wydania to:

- Przejście ze stosowania pakietu **sp** na pakiety **sf** i **stars**
- Dodanie nowych rycin i poprawienie istniejących
- Dodanie podpisów pod rycinami
- Zaktualizowanie źródeł wiedzy
- Poprawienie sekcji o transformacjach danych
- Dodanie sekcji o rozgrupowaniu danych

Wszystkie zaprezentowane metody i analizy zawierają również kod w języku R. Skrypt został stworzony w R (R Core Team, 2020) z wykorzystaniem pakietów **bookdown** (Xie, 2021a), **rmarkdown** (Allaire et al., 2021), **knitr** (Xie, 2021b) oraz programu Pandoc[1]. Aktualna wersja skryptu znajduje się pod adresem https://bookdown.org/nowosad/Geostatystyka/.

[1] http://pandoc.org/

Jeżeli używasz skryptu, zacytuj go jako:

- Nowosad, J., (2021). Geostatystyka w R. Poznań: Space A. ISBN 978-83-953296-2-3. Online: https://bookdown.org/nowosad/Geostatystyka/

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

# 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 (Hijmans et al., 2020; Nychka et al., 2021; Wickham et al., 2020; Auguie, 2017; Pebesma and Graeler, 2021; Appelhans et al., 2021; Giraudoux, 2021; Nowosad, 2019; Silge et al., 2021; Pebesma, 2021a,b).

```
pakiety = c(
  "dismo", "fields", "ggplot2", "gridExtra", "gstat", "mapview",
  "pgirmess", "rcartocolor", "rsample", "sf", "stars"
)
```

Pakiety R używane w tym skrypcie można zainstalować za pomocą metapakietu **geostatbook** (Nowosad, 2021):

```
install.packages("remotes")
remotes::install_github("nowosad/geostatbook@3")
```

[2]https://www.r-project.org/
[3]https://www.rstudio.com/

## 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]

```
# install.packages("remotes")
remotes::install_github("nowosad/geostatbook@3")
```

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

[4]https://github.com/Nowosad/geostat_book/blob/master/dane3.zip?raw=true
[5]https://github.com/Nowosad/geostatbook

# 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. Rycina 1.1 przestawia uproszczoną ścieżkę postępowania geostatystycznego.

Punktem wyjścia analizy geostatystycznej jest posiadanie danych przestrzennych opisujących badane zjawisko, np. w **postaci punktowej** (rycina 1.2).

Dane należy poddać eksploracji w celu ich lepszego poznania, wyszukania relacji między zmiennymi, czy znalezienia potencjalnych błędów (rycina 1.3).

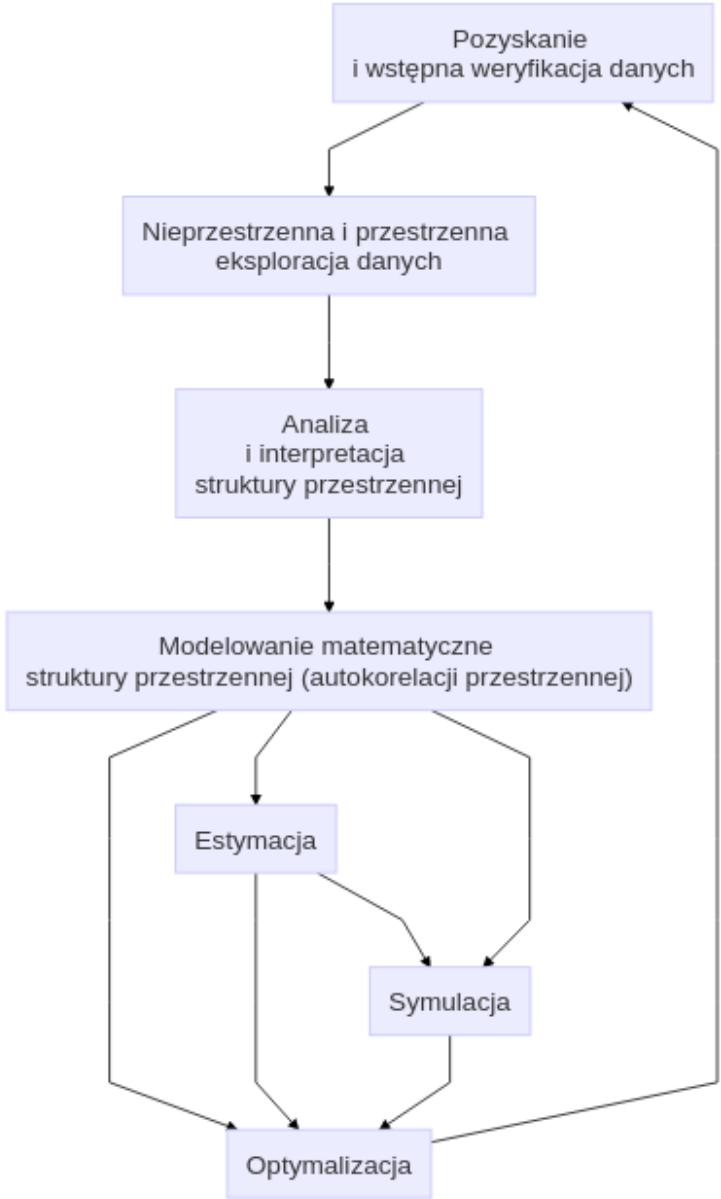
Na ich podstawie chcemy zrozumieć zmienność przestrzenną analizowanej cechy. Do tego może nam posłużyć wykres nazywany **semiwariogramem** (rycina 1.4).

**Semiwariogram** opisuje przestrzenną zmienność badanej cechy i za jego pomocą możemy stwierdzić jak to zjawisko zmienia się w przestrzeni. Dodatkowo za pomocą **mapy semiwariogramu** (rycina 1.5) możliwe jest stwierdzenie czy istnieją jakieś kierunki w których ta cecha zmienia się zmienia bardziej dynamicznie, a w których ta zmiana jest wolniejsza.

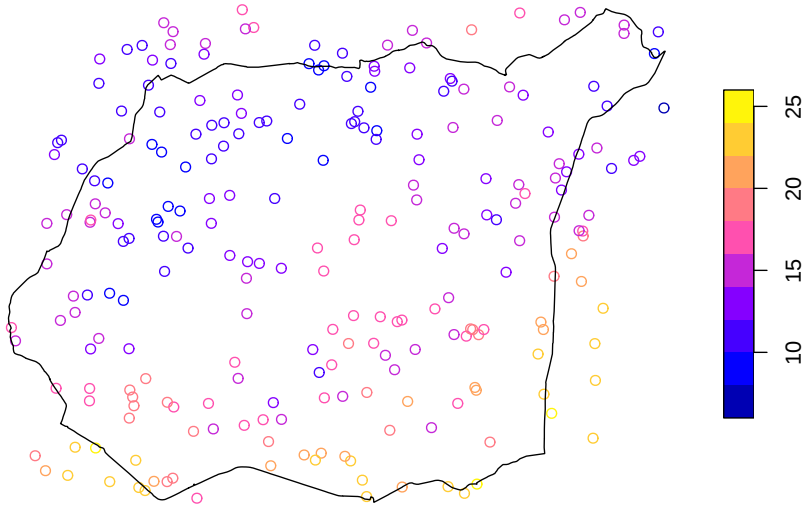
Następnie korzystając z wiedzy uzyskanej z semiwariogramu i mapy semiwariogramu, jesteśmy w stanie stworzyć **model semiwariogramu** (rycina 1.6).

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 (rycina 1.7).

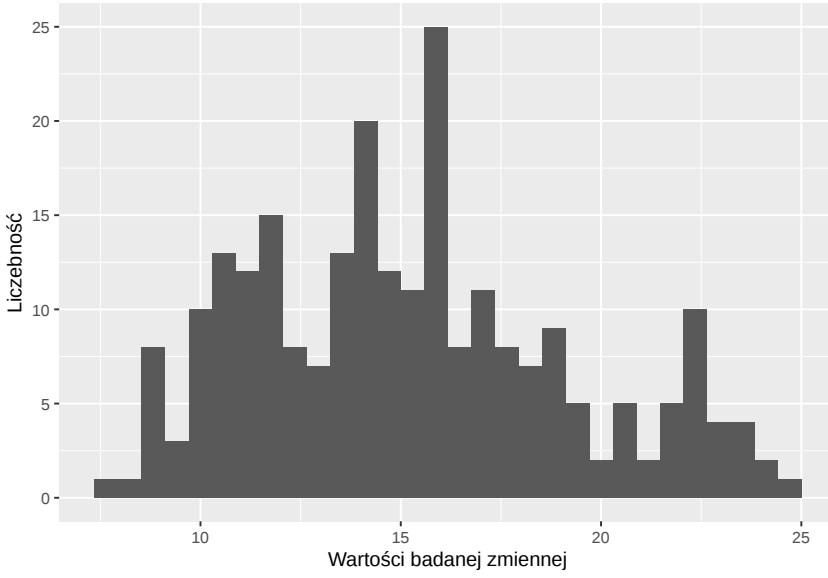
Rolą symulacji (rycina 1.8) jest natomiast generowanie równie prawdopodobne możliwości rozkładu badanej cechy.



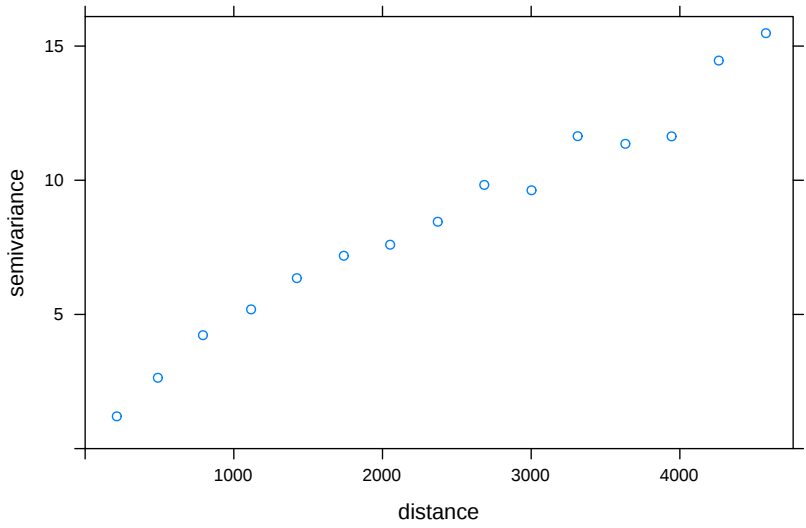

Rycina 1.1.: Uproszczona ścieżka postępowania geostatystycznego.



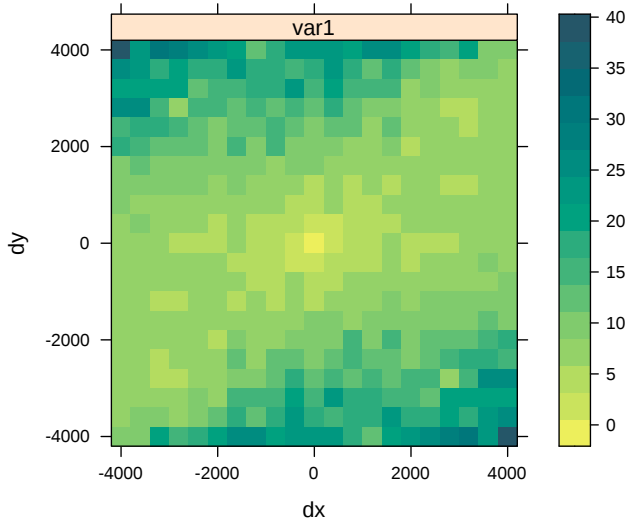

Rycina 1.2.: Przykładowe dane reprezentujące pomiary punktowe zmiennej numerycznej.



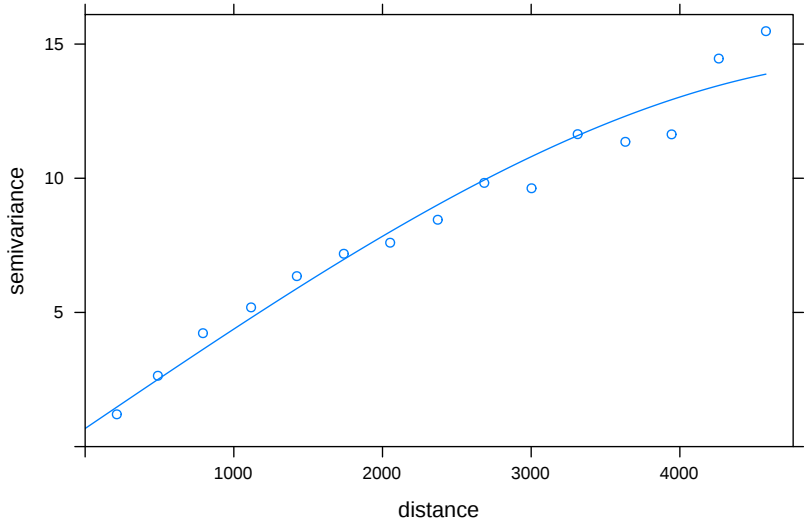

Rycina 1.3.: Rozkład wartości zmiennej numerycznej.



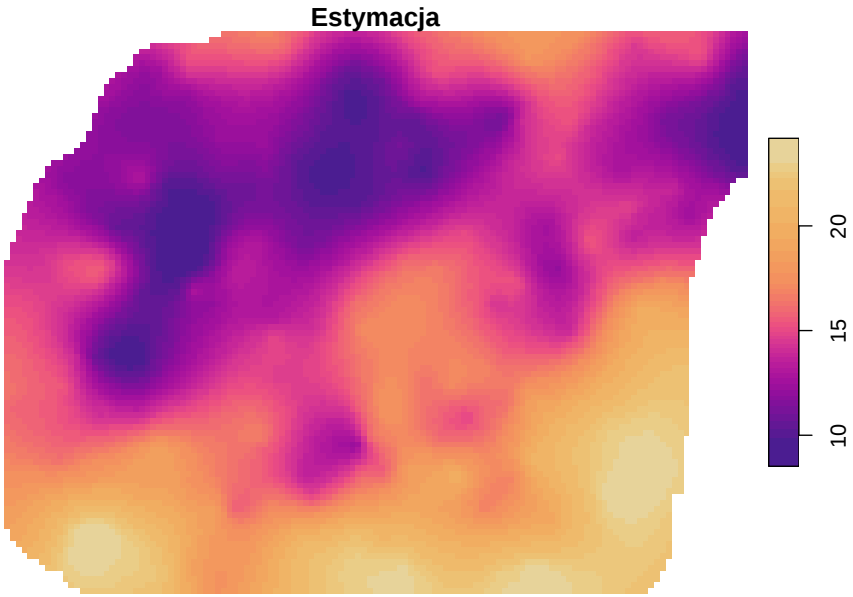

Rycina 1.4.: Wykres, nazywany semiwariogramem, reprezentujący niepodobieństwo wartości wraz z odległością.



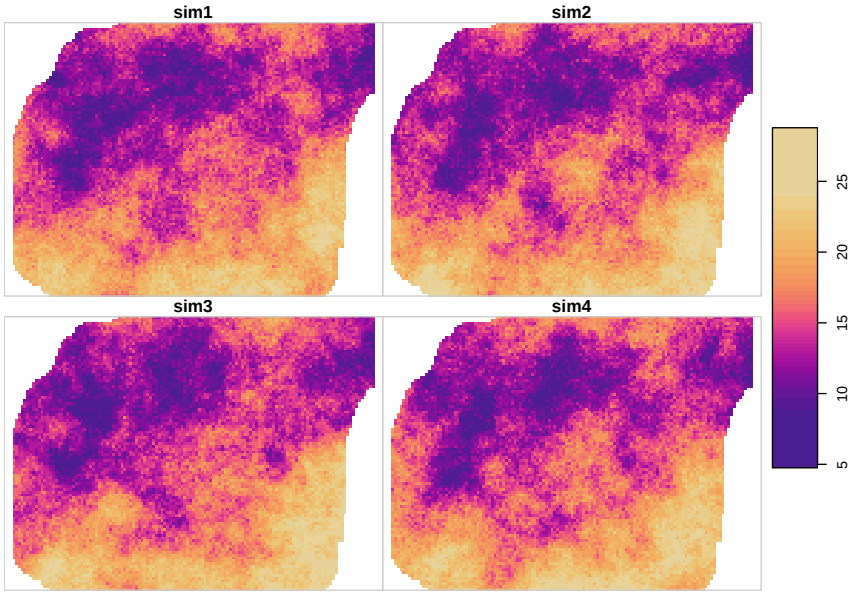

Rycina 1.5.: Mapa semiwariogramu reprezentująca niepodobieństwo wartości wraz z odległością i kierunkiem.




Rycina 1.6.: Model reprezentowany przez ciągłą linię naniesiony na semiwariogram.




Rycina 1.7.: Estymacja (oszacowanie) wartości badanej zmiennej dla całego obszaru.




Rycina 1.8.: Przykłady symulowanych wartości badanej zmiennej dla całego obszaru.

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. Do wbudowanych typów obiektów należą:

- 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)`
- Ramki danych (ang. *data.frame*) - to zbiór zmiennych (kolumn) oraz obserwacji (wierszy) zawierających różne typy danych
- Macierze (ang. *matrix*) - to zbiór kolumn oraz wierszy zawierających wektory jednego typu
- Listy (ang. *list*) - to sposób przechowywania obiektów o różnych typach i różnej długości

Dokładniejsze wyjaśnienie powyższych typów obiektów można znaleźć w rozdziałach 5 i 7 książki Elementarz programisty: Wstęp do programowania używając R[1].

### 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:

- Obsługa danych przestrzennych - **sf**, **stars**, **raster**, **terra**, **sp**, **rgdal**, **rgeos**
- Geostatystyka - **gstat**, **geoR**

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

[1] https://nowosad.github.io/elp/

### 2.1.3. Reprezentacja danych przestrzennych

Dane przestrzenne mogą być reprezentowane w R poprzez wiele różnych klas obiektów z użyciem różnych pakietów R. Przykładowo dane wektorowe mogą być reprezentowane poprzez obiekty klasy `Spatial*` z pakietu **sp** oraz obiekty klasy `sf` z pakietu **sf**.

Pakiet **sf** oparty jest o standard OGC o nazwie Simple Features. Łączy on możliwości zarówno pakietu **sp** jak i pakietów **rgdal** i **rgeos**. Większość jego funkcji zaczyna się od prefiksu `st_`. Ten pakiet definiuje klasę obiektów `sf`, która jest sposobem reprezentacji przestrzennych danych wektorowych. Pozwala on na stosowanie dodatkowych typów danych wektorowych (np. poligon i multipoligon to dwie oddzielne klasy), łatwiejsze przetwarzanie danych, oraz obsługę przestrzennych baz danych takich jak PostGIS. Pakiet **sf** jest używany przez kilkadziesiąt dodatkowych pakietów R, jednak wiele metod i funkcji nadal wymaga obiektów klasy **sp**. Porównanie funkcji dostępnych w pakietach **sp** i **sf** można znaleźć pod adresem https://github.com/r-spatial/sf/wiki/Migrating. Więcej o pakiecie **sf** można przeczytać w rozdziale drugim[2] książki Geocomputation with R.

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

Pakiet **stars** (*scalable, spatiotemporal tidy arrays*) pozwala na obsługę różnorodnych danych przestrzennych i czasoprzestrzennych reprezentowanych jako kostka danych (ang. *data cubes*). Obejmuje to, między innymi, regularne i nieregularne dane rastrowe, ale także czasoprzestrzenne dane wektorowe. Ten pakiet zapewnia klasy i metody do wczytywania, przetwarzania, wizualizacji czy zapisywania kości danych. Posiada on także wbudowaną integrację z pakietem **sf**.

### 2.1.4. GDAL/OGR

Pakiety **sf** czy **stars** wykorzystują zewnętrzne biblioteki GDAL/OGR[3] w R do wczytywania i zapisywania danych przestrzennych. GDAL to biblioteka zawierająca funkcje służące do odczytywania i zapisywania danych w formatach rastrowych, a OGR to biblioteka służąca to odczytywania i zapisywania danych w formatach wektorowych.

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

### 2.1.5. PROJ

Dane przestrzenne powinny być zawsze powiązane z układem współrzędnych - w ten sposób możliwe jest określenie gdzie są one położone, a także jak wykonywane są na nich operacje przestrzenne. Biblioteka PROJ[4] jest używana przez pakiety **sf** i **stars** do określania i konwersji układów współrzędnych. Układy współrzędnych mogą być opisywane na szereg sposobów, np. poprzez systemy kodów. Najpopularniejszym z nich jest system kodów EPSG (ang. *European Petroleum Survey Group*), który pozwala on na łatwe identyfikowanie układów współrzędnych. Przykładowo, układ PL 1992 może być określony jako `"EPSG:2180"`. Taki sposób określania układów współrzędnych wymaga jedynie znajomości odpowiedniego numery dla danego obszaru.

Dane przestrzenne przechowują też opis tego układu współrzędnych w bardziej złożny sposób nazwany WKT2. Zawiera on opis elementów związanych z konkretnym układem współrzędnych, takich jak jednostka, elipsoida odwzorowania, itd.

Warto też wiedzież, że przez wiele lat główną rolę pełniła reprezentacja **proj4string**, np. `"+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"`, która określała własności układu współrzędnych. Współcześnie jednak ta reprezentacja nie jest zalecana do użycia.

Strona http://www.spatialreference.org/ zawiera bazę danych układów współrzędnych.

### 2.1.6. Układy współrzędnych

Dane przestrzenne mogą być reprezentowane używając układów współrzędnych geograficznych lub prostokątnych płaskich.

W układzie współrzędnych geograficznych proporcje pomiędzy współrzędną oznaczającą długość geograficzną (X) a oznaczającą szerokość geograficzną (Y) nie są równe 1:1. Realna wielkość oczka siatki (jego powierzchnia) jest zmienna, a jednostka mapy jest abstrakcyjna w tych układach. W efekcie nie pozwala to na proste określanie odległości czy powierzchni. Powyższe cechy układów współrzędnych geograficznych powodują, że do większości algorytmów w geostatystyce wykorzystywane są układy współrzędnych prostokątnych płaskich.

Układy współrzędnych prostokątnych płaskich określane są w miarach liniowych (np. metrach). W tych układach proporcje między współrzędną X a Y są równe 1:1, a wielkość oczka jest stała.

[4] https://proj.org/

### 2.1.7. GEOS

Pakiet **sf** używa biblioteki GEOS[5] do wykonywania operacji 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 o rozszerzeniu `.csv`, dane wektorowe z plików `.shp`, dane rastrowe z plików w formacie GeoTIFF, oraz bazy danych przestrzennych z plików rozszerzeniu `.gpkg`.

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

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

```
dane_punktowe = read.csv("dane/punkty.csv")
```

```
head(dane_punktowe)
```

```
##       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. `dane_punktowe`) jest reprezentowany za pomocą klasy nieprzestrzennej `data.frame`. Aby obiekt został przetworzony do klasy przestrzennej, konieczne jest określenie które kolumny zawierają informacje o współrzędnych; w tym wypadku współrzędne znajduję się w kolumnach `x` oraz `y`. Nadanie układu współrzędnych odbywa się poprzez funkcję `st_as_sf()`.

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

```
library(sf)
dane_punktowe_sf = st_as_sf(dane_punktowe, coords = c("x", "y"))
dane_punktowe_sf
```

```
## Simple feature collection with 248 features and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 745592.5 ymin: 712642.4 xmax: 756967.8 ymax: 721238.7
## CRS:           NA
## First 10 features:
##        srtm clc      temp      ndvi      savi                   geometry
## 1  175.7430   1 13.852222 0.6158061 0.4189449   POINT (750298 716731.6)
## 2  149.8111   1 15.484209 0.5558816 0.3794864 POINT (753482.9 717331.4)
## 3  272.8583  NA 12.760814 0.6067462 0.3745572   POINT (747242.5 720589)
## 4  187.2777   1 14.324648 0.3756170 0.2386246 POINT (755798.9 718828.1)
## 5  260.1366   1 15.908549 0.4598393 0.3087599 POINT (746963.5 717533.5)
## 6  160.1416   2  9.941118 0.5600288 0.3453627 POINT (756801.6 720474.1)
## 7  192.9223   2 13.514751 0.6009588 0.3737249   POINT (752698 718623.6)
## 8  234.9989   1 12.919225 0.4979195 0.2978503 POINT (749399.8 716955.2)
## 9  228.7312   1 19.247132 0.3819634 0.2077981   POINT (748766.6 713889)
## 10 240.5474   2 10.529105 0.5795159 0.3805268   POINT (749290.4 718861)
```

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

```
dane_punktowe_sf = st_set_crs(dane_punktowe_sf, value = 2180)
```

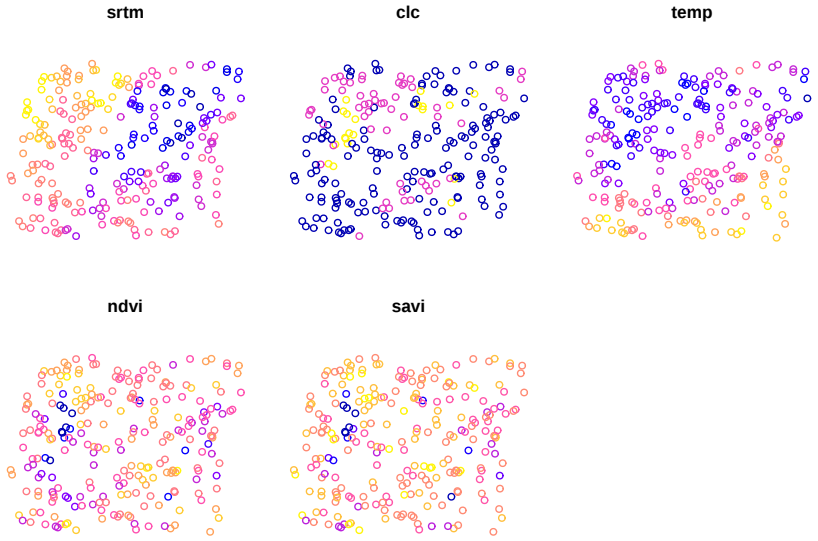
Proste wizualizacje uzyskanych danych klasy przestrzennej, np. `sf` czy `stars`, można uzyskać za pomocą funkcji `plot()` (rycina 2.1 i 2.2).

```
plot(dane_punktowe_sf)
```

### 2.2.2. Dane wektorowe

Dane wektorowe (np. w formacie GeoPackage czy ESRI Shapefile) można odczytać za pomocą funkcji `read_sf()` z pakietu **sf**. Przyjmuje ona co najmniej jeden argument - `dsn`, który określa główny plik w którym znajdują się dane.

```
library(sf)
granica_sf = read_sf(dsn = "dane/granica.gpkg")
# plot(granica_sf)
```




Rycina 2.1.: Efekt działania funkcji plot na obiekcie klasy sf.

### 2.2.3. Dane rastrowe

Istnieje kilka sposobów odczytu danych rastrowych w R. Do najpopularniejszych należą funkcje `readGDAL()` z pakietu **rgdal** oraz `raster()` z pakietu **raster**. W poniższym przykładzie użyjemy funkcji `read_stars()` z pakietu **stars**. Należy w niej jedynie podać ścieżkę do pliku rastrowego.

```
library(stars)
siatka_stars = read_stars("dane/siatka.tif")
siatka_stars
```

```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##             Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
## siatka.tif     0       0      0    0       0    0 1270
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92 FALSE   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92 FALSE   NULL [y]
```

```
plot(siatka_stars)
```


Rycina 2.2.: Efekt działania funkcji plot na obiekcie klasy sf.

## 2.3. Przeglądanie danych przestrzennych

### 2.3.1. Struktura obiektu

Podstawowe informacje o obiekcie można uzyskać poprzez wpisanie jego nazwy:

```
dane_punktowe_sf
```

```
## Simple feature collection with 248 features and 5 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 745592.5 ymin: 712642.4 xmax: 756967.8 ymax: 721238.7
## Projected CRS: ETRS89 / Poland CS92
## First 10 features:
##        srtm clc      temp      ndvi      savi                   geometry
## 1  175.7430   1 13.852222 0.6158061 0.4189449   POINT (750298 716731.6)
## 2  149.8111   1 15.484209 0.5558816 0.3794864 POINT (753482.9 717331.4)
## 3  272.8583  NA 12.760814 0.6067462 0.3745572   POINT (747242.5 720589)
## 4  187.2777   1 14.324648 0.3756170 0.2386246 POINT (755798.9 718828.1)
## 5  260.1366   1 15.908549 0.4598393 0.3087599 POINT (746963.5 717533.5)
## 6  160.1416   2  9.941118 0.5600288 0.3453627 POINT (756801.6 720474.1)
```

```
## 7  192.9223   2 13.514751 0.6009588 0.3737249   POINT (752698 718623.6)
## 8  234.9989   1 12.919225 0.4979195 0.2978503 POINT (749399.8 716955.2)
## 9  228.7312   1 19.247132 0.3819634 0.2077981   POINT (748766.6 713889)
## 10 240.5474   2 10.529105 0.5795159 0.3805268   POINT (749290.4 718861)
```

Obiekty `sf` są zbudowane z tabeli (*data frame*) wraz z dodatkową kolumną zawierającą geometrie (często nazywaną `geometry` lub `geom`) oraz szeregu atrybutów przestrzennych. Strukturę obiektu `sf` można sprawdzić za pomocą funkcji `str()`:

```
str(dane_punktowe_sf)
```

```
## Classes 'sf' and 'data.frame':   248 obs. of  6 variables:
##  $ srtm    : num  176 150 273 187 260 ...
##  $ clc     : int  1 1 NA 1 1 2 2 1 1 2 ...
##  $ temp    : num  13.9 15.5 12.8 14.3 15.9 ...
##  $ ndvi    : num  0.616 0.556 0.607 0.376 0.46 ...
##  $ savi    : num  0.419 0.379 0.375 0.239 0.309 ...
##  $ geometry:sfc_POINT of length 248; first list element:  'XY' num  750298 716732
##  - attr(*, "sf_column")= chr "geometry"
## - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA
##   ..- attr(*, "names")= chr [1:5] "srtm" "clc" "temp" "ndvi" ...
```

Obiekty `stars` mogą przyjmować różną formę w zależności od tego czy reprezentują one dane wektorowe czy rastrowe oraz tego ile wymiarów posiadają te dane. Przykładowy obiekt `siatka_stars` posiada dwa wymiary (`x` i `y`) oraz jeden atrybut (`siatka.tif`).

```
siatka_stars
```

```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##             Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
## siatka.tif     0       0      0    0       0    0 1270
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92 FALSE   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92 FALSE   NULL [y]
```

Jest on zbudowany jako lista zawierająca dwuwymiarową macierz wraz ze zbiorem metadanych (można to sprawdzić za pomocą funkcji `str(siatka_stars)`).

### 2.3.2. Tabla atrybutów

Funkcja `st_drop_geometry()` pozwala na pozbycie się informacji przestrzennych z obiektu klasy `sf` i uzyskanie jedynie obiektu klasy `data.frame` zawierającego nieprzestrzenne informacje o kolejnych punktach w tym zbiorze.

```
punkty_df = st_drop_geometry(dane_punktowe_sf)
head(punkty_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
```

Przetworzenie obiektu klasy `stars` na `data.frame` odbywa się natomiast używając funkcji `as.data.frame()`.

```
siatka_df = as.data.frame(siatka_stars)
head(siatka_df)
```

```
##          x        y siatka.tif
## 1 745586.7 721211.2         NA
## 2 745676.7 721211.2         NA
## 3 745766.7 721211.2         NA
## 4 745856.7 721211.2         NA
## 5 745946.7 721211.2         NA
## 6 746036.7 721211.2         NA
```

### 2.3.3. Współrzędne

Funkcja `st_coordinates()` pozwala na wydobycie współrzędnych zarówno z obiektu punktowego klasy `sf` jak i siatki klasy `stars`.

```
# sf
xy = st_coordinates(dane_punktowe_sf)
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
```

```
# stars
siatka_xy = st_coordinates(siatka_stars)
head(siatka_xy)
```

```
##          x        y
## 1 745586.7 721211.2
## 2 745676.7 721211.2
## 3 745766.7 721211.2
## 4 745856.7 721211.2
## 5 745946.7 721211.2
## 6 746036.7 721211.2
```

### 2.3.4. Obwiednia

Funkcja `st_bbox()` określa zasięg przestrzenny danych w jednostkach mapy.

```
st_bbox(dane_punktowe_sf)
```

```
##     xmin     ymin     xmax     ymax
## 745592.5 712642.4 756967.8 721238.7
```

### 2.3.5. Układ współrzędnych

Funkcja `st_crs()` wyświetla definicję układu współrzędnych.

```
st_crs(dane_punktowe_sf)
```

```
## Coordinate Reference System:
##   User input: EPSG:2180
##   wkt:
## PROJCRS["ETRS89 / Poland CS92",
##     BASEGEOGCRS["ETRS89",
##         DATUM["European Terrestrial Reference System 1989",
```

```
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4258]],
##     CONVERSION["Poland CS92",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",19,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9993,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",-5300000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["northing (x)",north,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["easting (y)",east,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["Poland"],
##         BBOX[49,14.14,55.93,24.15]],
##     ID["EPSG",2180]]
```

## 2.4. Przetwarzanie danych przestrzennych

### 2.4.1. Łączenie danych rastrowych

Połączenie danych rastrowych pochodzących z kilku plików może odbyć się poprzez wczytanie ich jako osobnych obiektów klasy `stars` a następnie połączenie ich używając funkcji `c()`.

```
srtm_stars = read_stars("dane/srtm.tif")
clc_stars = read_stars("dane/clc.tif")
dwie_stars = c(srtm_stars, clc_stars)
```

W efekcie uzyskiwany jest obiekt posiadający dwa atrybuty odpowiadające wczytanym plikom.

```
dwie_stars
```

```
## stars object with 2 dimensions and 2 attributes
## attribute(s):
##               Min.  1st Qu.   Median      Mean  3rd Qu.     Max. NA's
## srtm.tif  143.6559 186.4765 216.4929 212.21245 235.2941 288.8382 1242
## clc.tif     1.0000   1.0000   1.0000   1.42126   2.0000   4.0000 1270
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92 FALSE   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92 FALSE   NULL [y]
```

### 2.4.2. Wydobywanie wartości z rastra

Posiadając dwa obiekty, jeden punktowy a drugi rastrowy, dotyczące tego samego obszaru możliwe jest wydobycie wartości dla punktów używając funkcji `st_join()`.

```
dane_punktowe_sf2 = st_join(dane_punktowe_sf, st_as_sf(dwie_stars))
```

## 2.5. Eksport danych

### 2.5.1. Zapisywanie danych wektorowych

R pozwala również na zapisywanie danych przestrzennych. W przypadku zapisu danych wektorowych za pomocą funkcji `write_sf()` konieczne jest podanie ścieżki i nazwy zapisywanego obiektu wraz z rozszerzeniem (np. `dane/granica.gpkg`).

```
write_sf(granica_sf, dsn = "dane/granica.gpkg")
```

### 2.5.2. Zapisywanie danych rastrowych

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

```
write_stars(siatka_stars, dsn = "dane/nowa_siatka.tif")
```

## 2.6. Wizualizacja danych 2D

Do wizualizacji danych przestrzennych w R służy co najmniej kilkanaście różnych pakietów. Poniżej pokazane są przykłady standardowej funkcji `plot()`. Więcej o wizualizacji danych przestrzennych można przeczytać w rozdziale Making maps with R[6] książki Geocomputation with R.

### 2.6.1. Dane punktowe

Funkcja `plot()` idealnie nadaje się do szybkiego przyjrzenia się, np. rodzajowi próbkowania danych. Domyślnie wyświetla ona szereg map - po jednej dla każdej zmiennej (rycina 2.3).

```
plot(dane_punktowe_sf)
```

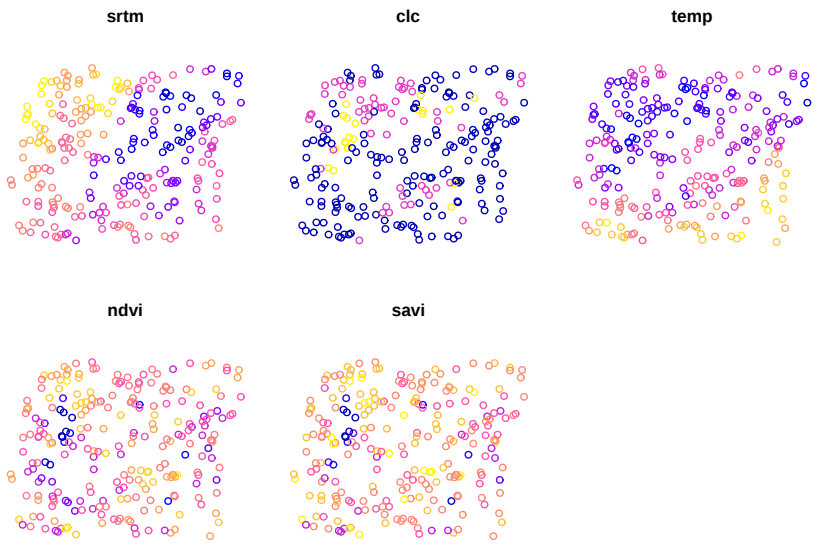
Do wyświetlenia tylko geometrii (np. punktów) bez atrybutów służy funkcja `plot()` połączona z funkcją `st_geometry()` (rycina 2.4).

```
plot(st_geometry(dane_punktowe_sf))
```

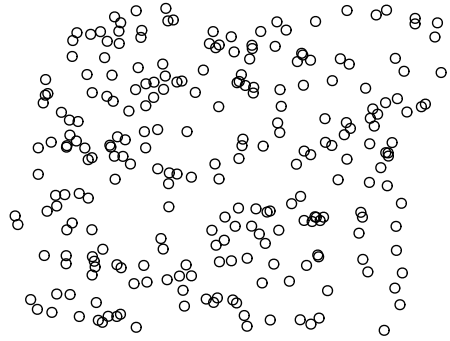
Aby wyświetlić tylko jedną zmienną należy ją zadeklarować poprzez nawias kwadratowy. Poniżej można zobaczyć przykłady dla zmiennej `srtm` (rycina 2.5).

```
plot(dane_punktowe_sf["srtm"])
```

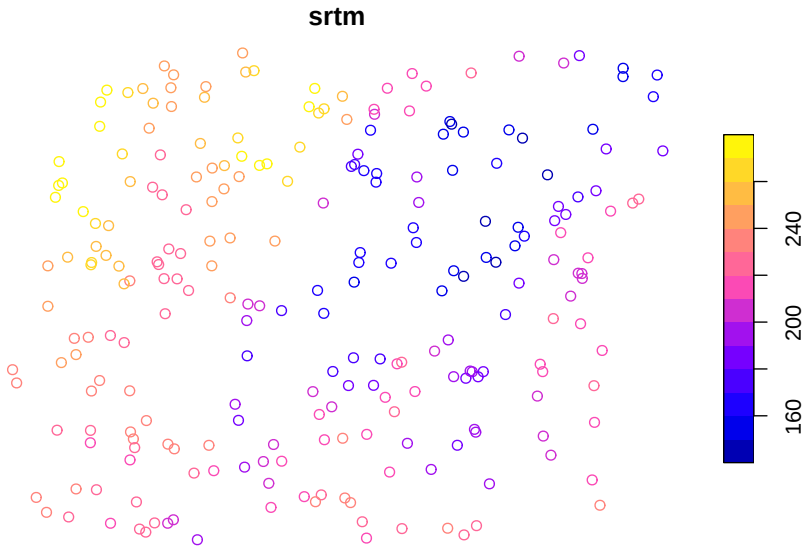
[6] https://geocompr.robinlovelace.net/adv-map.html




Rycina 2.3.: Użycie funkcji plot na danych punktowych klasy sf.



Rycina 2.4.: Użycie funkcji plot do wyświetlenia geometrii obiektu klasy sf.




Rycina 2.5.: Użycie funkcji plot do wyświetlenia zmiennej numerycznej obiektu klasy sf.

### 2.6.2. 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 kategoryzowanej (`as.factor()`). Następnie można je wyświetlić za pomocą funkcji `plot()` (rycina 2.6).

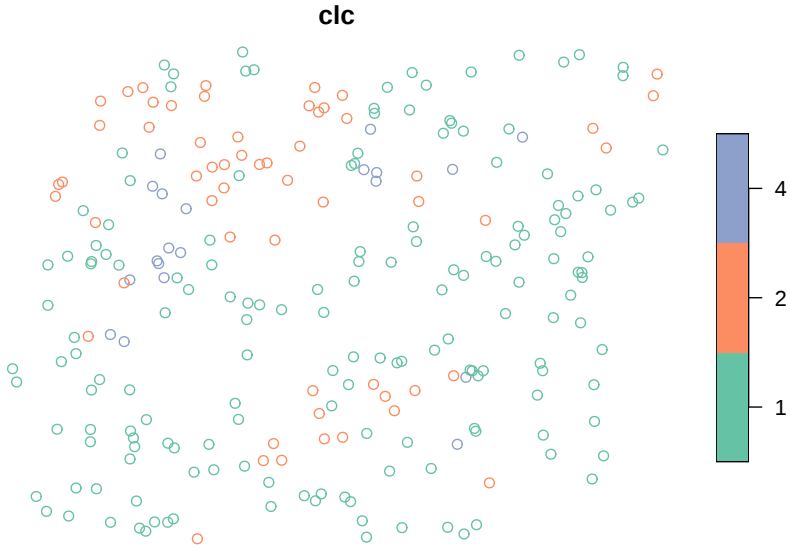
```
dane_punktowe_sf$clc = as.factor(dane_punktowe_sf$clc)
plot(dane_punktowe_sf["clc"])
```

### 2.6.3. Rastry

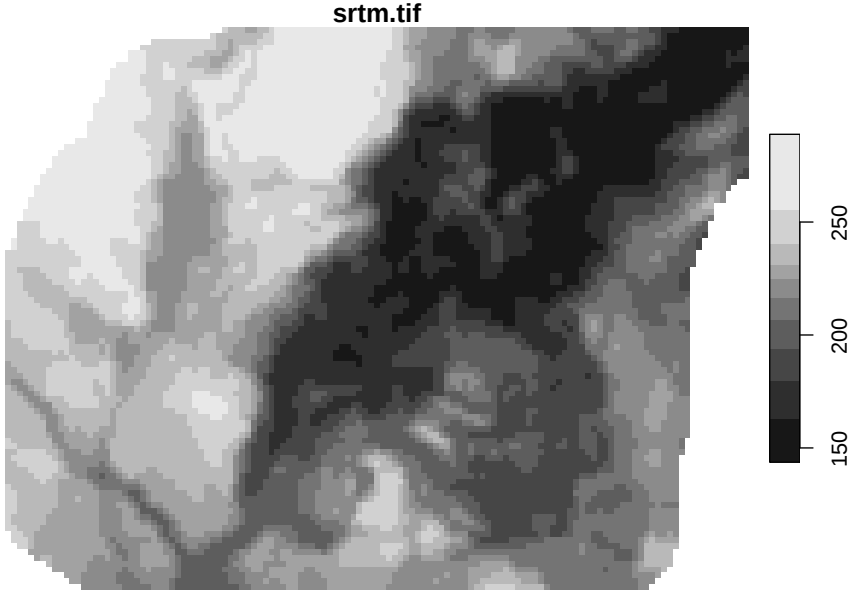
Wyświetlanie danych w formacie rastrowym może również odbyć się z użyciem funkcji `plot()`. Domyślnie wyświetla ona pierwszą zmienną (lub warstwę) znajdującą się w obiekcie klasy `stars` (rycina 2.7).

```
plot(dwie_stars)
```

Zmiana kolorów mapy odbywa się poprzez podanie wektora kolorów (palety) w argumencie `col` (rycina 2.8).

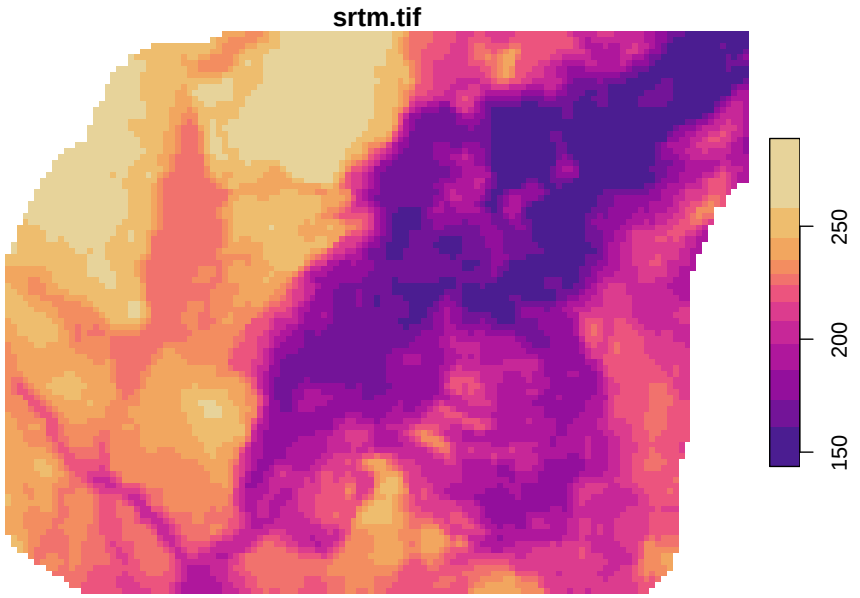



Rycina 2.6.: Użycie funkcji plot do wyświetlenia zmiennej kategoryzowanej obiektu klasy sf.




Rycina 2.7.: Użycie funkcji plot do wyświetlenia pierwszej zmiennej obiektu klasy stars.

```
paleta = hcl.colors(12, palette = "ag_Sunset")
plot(dwie_stars, col = paleta)
```




Rycina 2.8.: Użycie funkcji plot do wyświetlenia pierwszej zmiennej obiektu klasy stars wraz z zadaną paletą kolorów.

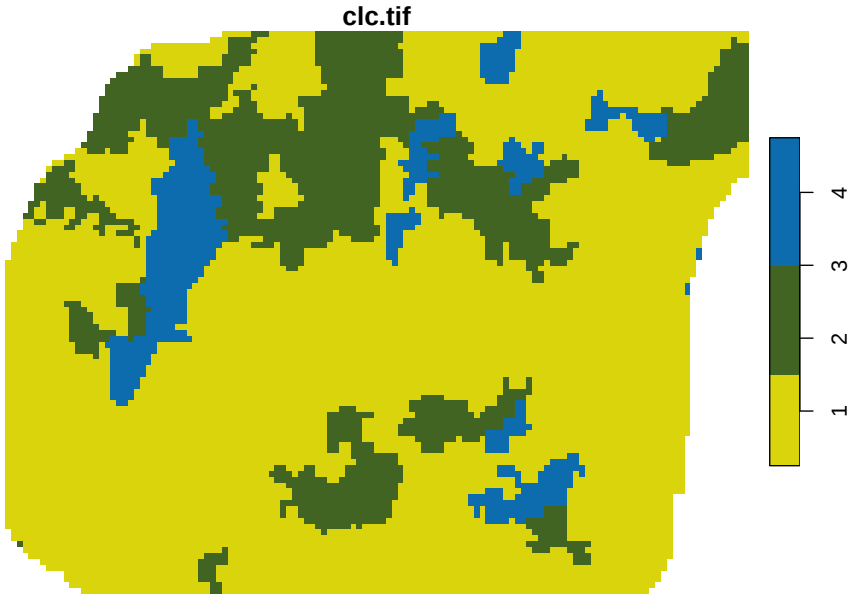
Aby wyświetlić tylko wybraną zmienną należy ją zadeklarować poprzez indeksowanie. Opiera się to o kilka indeksów, np. dla obiektu `siatka_stars` są to `[indeks atrybutu, indeksy kolumn, indeksy wierszy]`[7]. Przykładowo indeks `[2, , ]` pozwala na wyświetlenie drugiego atrybutu. Możliwe jest tutaj również użycie uproszczonej formy `dwie_stars[2]` (rycina 2.9).

```
plot(dwie_stars[2], col = c("#d9d40c", "#416422", "#0c6cae"))
```

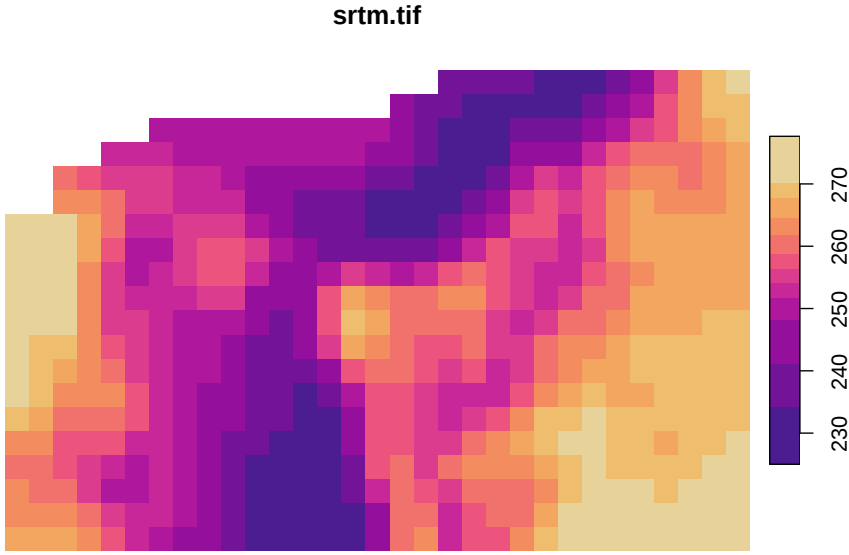
Indeksowanie pozwala też na wyświetlenie fragmentu danych rastrowych. Na przykład, indeks `[1, 2:50, 1:20]` pozwala na wyświetlenie pierwszej zmiennej, ale tylko dla kolumn od 20 do 50 i wierszy od 1 do 20 (rycina 2.10).

```
plot(dwie_stars[1, 20:50, 1:20], col = paleta)
```

[7] Liczba indeksów zależy od złożoności obiektu klasy `stars`. Przykładowo dla obiektów posiadających trzy wymiary, np. wielokanałowe dane satelitarne, mogą to być `[indeks atrybutu, indeksy kolumn, indeksy wierszy, indeksy kanału]`.




Rycina 2.9.: Użycie funkcji plot do wyświetlenia wybranej zmiennej obiektu klasy stars.




Rycina 2.10.: Użycie funkcji plot do wyświetlenia wybranej zmiennej obiektu klasy stars dla wybranego zasięgu przestrzennego.

## 2.7. 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.
4. Wczytaj ten nowy plik w formacie ESRI Shapefile do R. Przejrzyj ten obiekt. Jaki typ geometrii on przechowuje? Ile obiektów on zawiera? Ile ma on atrybutów (zmiennych)? Jaki ma on układ współrzędnych?
5. Nadaj powyższemu obiektowi układ współrzędnych PL1992 (`EPSG:2180`).
6. Wczytaj dane z pliku `srtm.tif` do R jako obiekt `srtm`. Jakiego rodzaju jest wynikowy obiekt? Jaki typ danych on przechowuje? Jakie ma wymiary? Ile ma on atrybutów (zmiennych)? Jaki ma on układ współrzędnych? Stwórz prostą mapę z nowo utworzonego obiektu.
7. Dla punktów z obiektu `ndvi_p` wydobądź wartości z obiektu `srtm`.
8. Stwórz mapę pokazującą punkty z obiektu `ndvi_p`, gdzie kolory będą obrazowały wartości wyciągnięte z obiektu `srtm`. W tle tej mapy przedstaw obiekt `srtm`. (Dodatkowo: spróbuj do tego użyć pakietu **tmap** lub **ggplot2**).

# 3. Eksploracyjna analiza danych nieprzestrzennych

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

```
library(sf)
library(stars)
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 `sf` wyświetla ona szereg ważnych informacji.

```
str(punkty)
```

```
## Classes 'sf' and 'data.frame':   242 obs. of  6 variables:
##  $ srtm    : num  176 150 187 260 160 ...
##  $ clc     : int  1 1 1 1 2 2 1 1 2 1 ...
##  $ temp    : num  13.85 15.48 14.32 15.91 9.94 ...
##  $ ndvi    : num  0.616 0.556 0.376 0.46 0.56 ...
##  $ savi    : num  0.419 0.379 0.239 0.309 0.345 ...
##  $ geometry:sfc_POINT of length 242; first list element:  'XY' num  750298 716732
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA
##    ..- attr(*, "names")= chr [1:5] "srtm" "clc" "temp" "ndvi" ...
##  - attr(*, "na.action")= 'omit' Named int [1:6] 3 90 110 115 140 197
##    ..- attr(*, "names")= chr [1:6] "3" "90" "110" "115" ...
```

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

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

```
##  num [1:242] 13.85 15.48 14.32 15.91 9.94 ...
```

## 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)
```

```
##       srtm            clc             temp             ndvi
##  Min.   :146.5   Min.   :1.000   Min.   : 7.883   Min.   :0.2024
##  1st Qu.:190.4   1st Qu.:1.000   1st Qu.:11.953   1st Qu.:0.4629
##  Median :217.4   Median :1.000   Median :14.937   Median :0.5141
##  Mean   :214.3   Mean   :1.483   Mean   :15.223   Mean   :0.5034
##  3rd Qu.:238.4   3rd Qu.:2.000   3rd Qu.:17.584   3rd Qu.:0.5712
##  Max.   :278.4   Max.   :4.000   Max.   :24.945   Max.   :0.6597
##       savi                 geometry
```

```
##  Min.   :0.0824   POINT        :242
##  1st Qu.:0.2923   epsg:2180    :  0
##  Median :0.3253   +proj=tmer...:  0
##  Mean   :0.3170
##  3rd Qu.:0.3591
##  Max.   :0.4404
```

### 3.3.2. Średnia i mediana

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

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

```
## [1] 14.93736
```

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

```
## [1] 15.22251
```

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] 7.882644
```

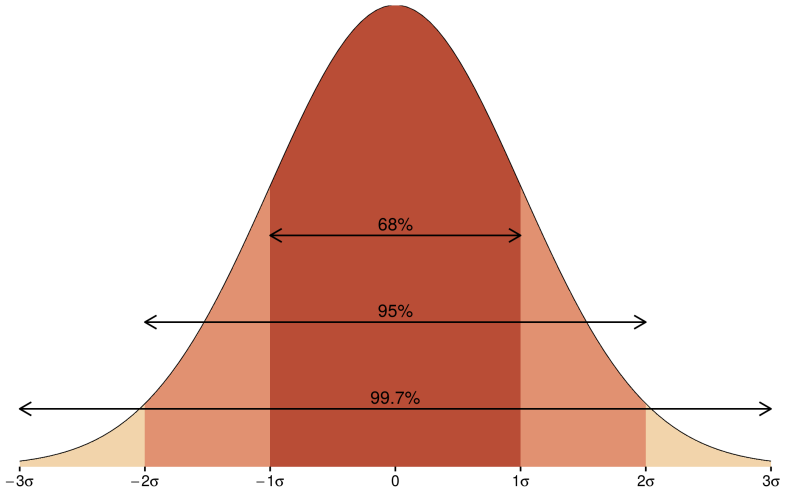
```
max(punkty$temp, na.rm = TRUE)
```

```
## [1] 24.94451
```

### 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 (rycina 3.1):

- 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




Rycina 3.1.: Rozrzut wartości w rozkładzie normalnym w zależności od odchylenia standardowego.

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

```
## [1] 3.954922
```

### 3.3.5. Liczebność grup

Dla zmiennych kategoryzowanych nie powinno wyliczać się powyższych statystyk. Możliwe jest natomiast określenie liczebności grup używając funkcji `table()`.

```
table(punkty$clc)
```

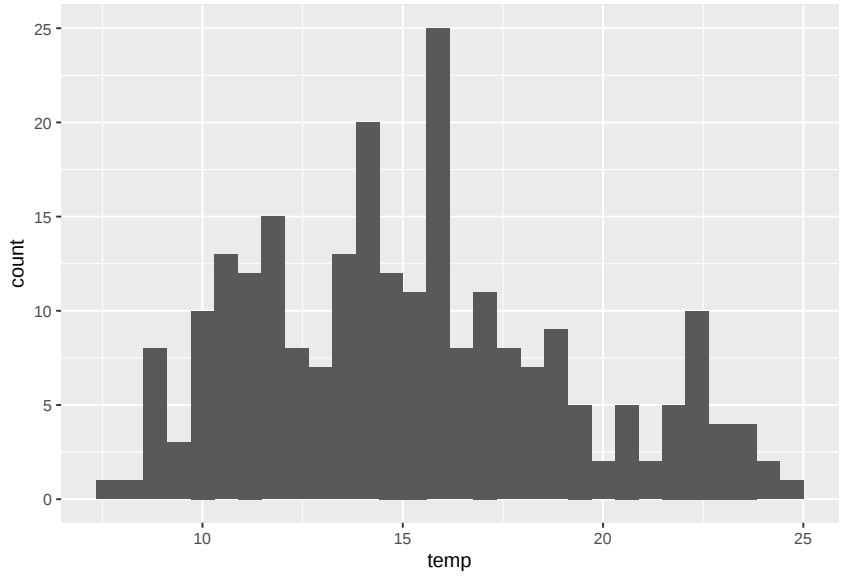
```
##
##   1   2   4
## 165  57  20
```

## 3.4. Wykresy

### 3.4.1. Histogram

Histogram należy do typów wykresów najczęściej używanych w eksploracyjnej analizie danych (rycina 3.2).

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



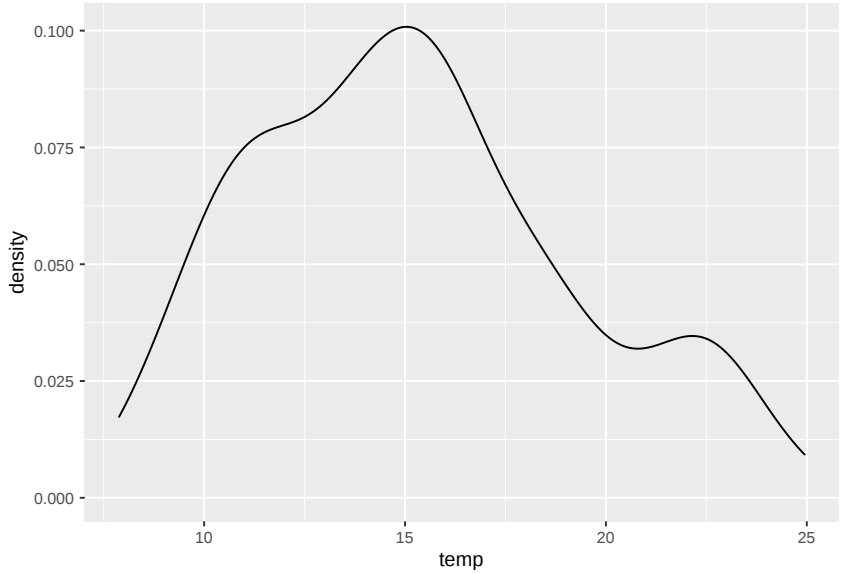

Rycina 3.2.: Histogram reprezentujący wartości zmiennej temp z obiektu punkty.

Jest on 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. Co ważne, 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

Podobną funkcję do histogramu spełnia estymator jądrowy gęstości (ang. *kernel density estimation*). Przypomina on wygładzony wykres histogramu i również służy graficznej reprezentacji rozkładu danych (rycina 3.3).

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



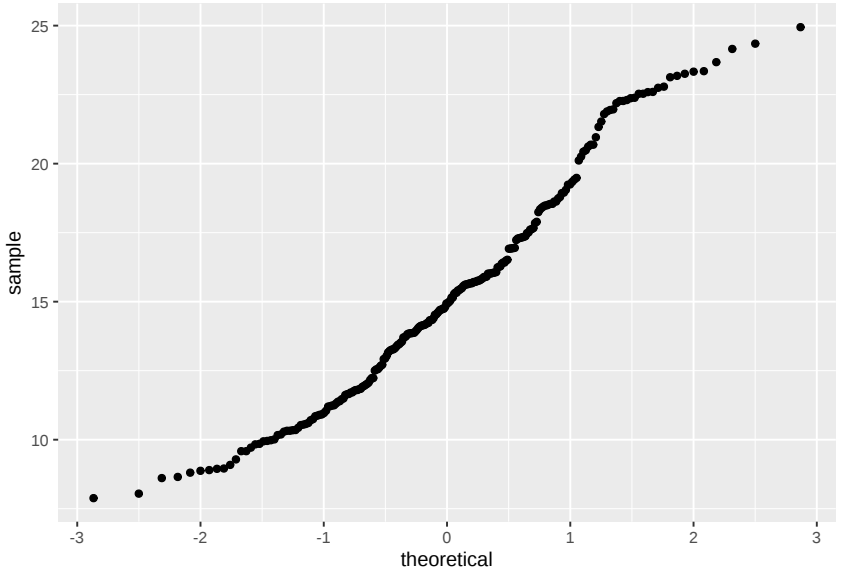

Rycina 3.3.: Rozkład wartości zmiennej temp z obiektu punkty.

### 3.4.3. Wykres kwantyl-kwantyl

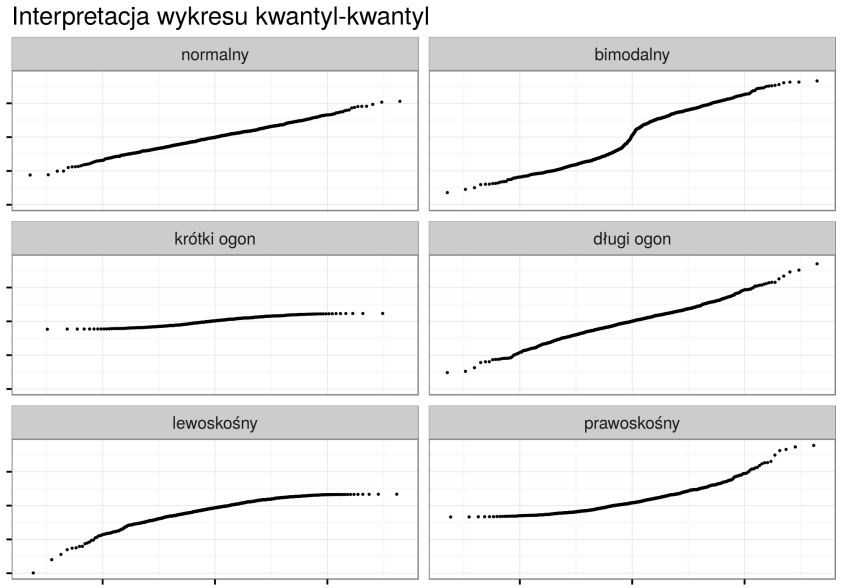
Wykres kwantyl-kwantyl (ang.*quantile-quantile*) ułatwia interpretację rozkładu danych (rycina 3.4).

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

Na poniższej rycinie można zobaczyć przykłady najczęściej spotykanych cech rozkładu danych w wykresach kwantyl-kwantyl (rycina 3.5).




Rycina 3.4.: Wykres kwantyl-kwantyl wartości zmiennej temp z obiektu punkty.

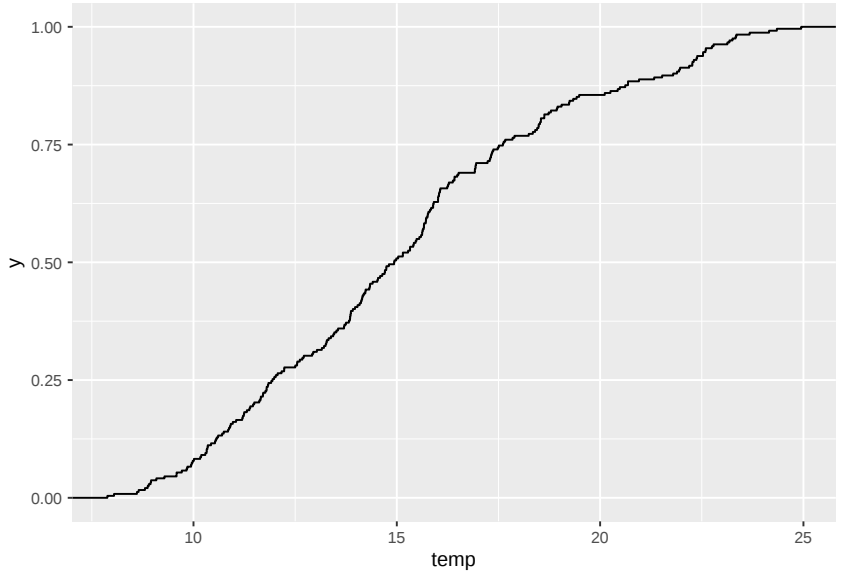



Rycina 3.5.: Interpretacja rozkładu danych w zależności od wyglądu wykresu kwantyl-kwantyl.

### 3.4.4. Dystrybuanta (CDF)

Dystrybuanta (ang. *cumulative distribution function* - CDF) wyświetla prawdopodobieństwo, że wartość zmiennej przewidywanej jest mniejsza lub równa określonej wartości (rycina 3.6).

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




Rycina 3.6.: Dystrybuanta zmiennej temp z obiektu punkty.

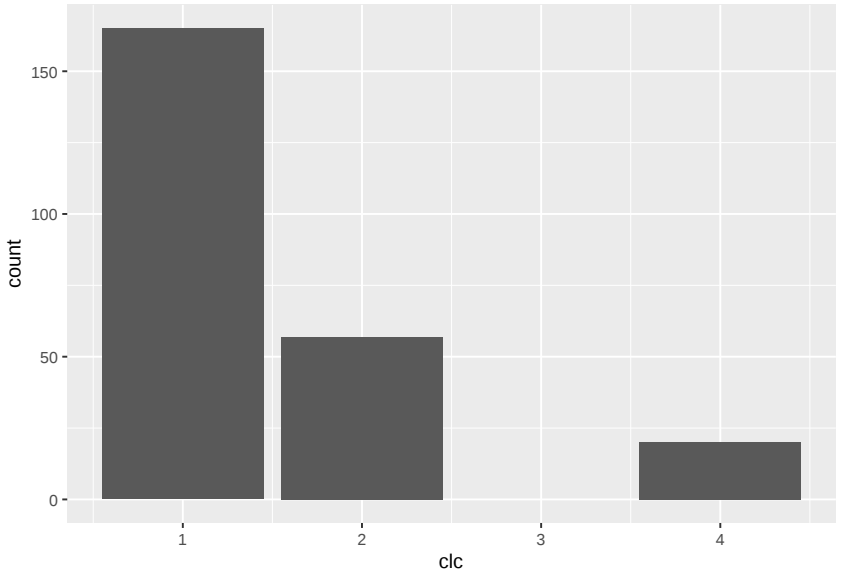
### 3.4.5. Wykres słupkowy

Wykres słupkowy (ang. *bar plot*) przedstawia liczebność kolejnych grup (rycina 3.7).

```
ggplot(punkty, aes(clc)) + geom_bar()
```

## 3.5. Porównanie zmiennych

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




Rycina 3.7.: Wykres słupkowy zmiennej clc z obiektu punkty.

### 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 wyniki 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()`.

### 3.5.2. Współczynnik korelacji

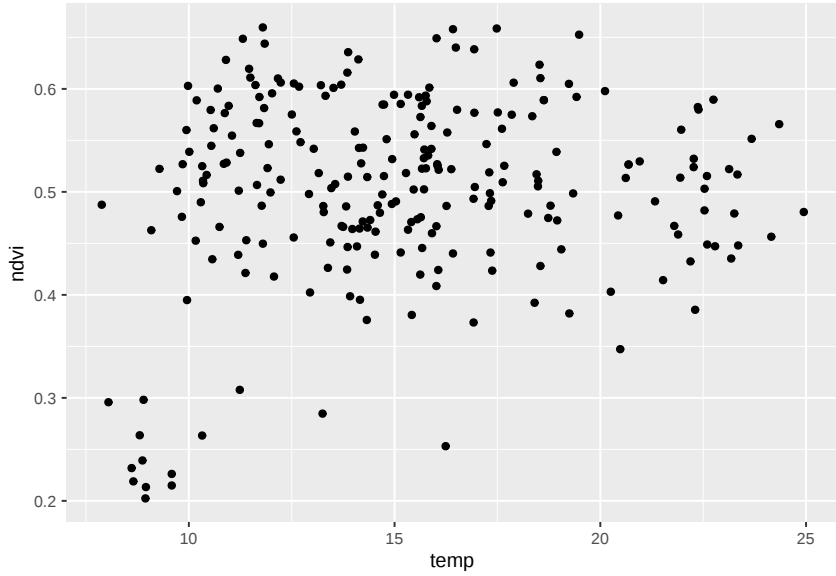
Współczynnik korelacji to unormowana miara zależności pomiędzy dwiema zmiennymi, przyjmująca wartości od -1 do 1. Ten współczynnik jest uzyskiwany poprzez podzielenie wartości kowariancji przez odchylenie standardowe wyników. Z racji unormowania nie jest ona uzależniona od

jednostki. Korelację można wyliczyć dzięki funkcji `cor()` (rycina 3.8). Działa ona zarówno w przypadku dwóch zmiennych numerycznych, jak i całego obiektu zawierającego zmienne numeryczne. Za pomocą argumentu `method` można również wybrać jedną z trzech dostępnych miar korelacji: `"pearson"` przeznaczoną dla zmiennych o rozkładzie normalnym oraz `"spearman"` i `"kendall"` używaną dla zmiennych o rozkładzie różnym od normalnego.

```
cor(punkty$temp, punkty$ndvi, method = "spearman")
```

```
## [1] 0.04499375
```

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




Rycina 3.8.: Wykres rozrzutu reprezentujący relację pomiędzy zmienną temp i ndvi z obiektu punkty.

Dodatkowo funkcja `cor.test()` służy do testowania istotności korelacji.

```
cor.test(punkty$temp, punkty$ndvi, method = "spearman")
```

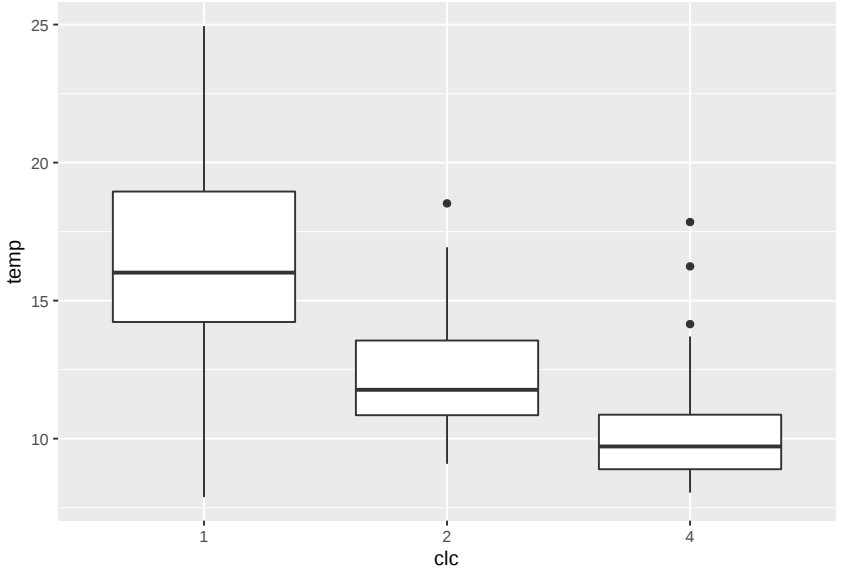
```
##
##  Spearman's rank correlation rho
##
```

```
## data:  punkty$temp and punkty$ndvi
## S = 2255764, p-value = 0.486
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## 0.04499375
```

### 3.5.3. Wykres pudełkowy

Wykres pudełkowy obrazuje pięć podstawowych statystyk opisowych oraz wartości odstające (rycina 3.9). 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, aes(x = clc, y = temp)) + geom_boxplot()
```




Rycina 3.9.: Wykres pudełkowy pokazujący relację pomiędzy zmienną clc i temp z obiektu punkty.

### 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 natomiast 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   1272   636.0   60.86 <0.0000000000000002 ***
## Residuals   239   2498    10.4
## ---
## 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()` (rycina 3.10).

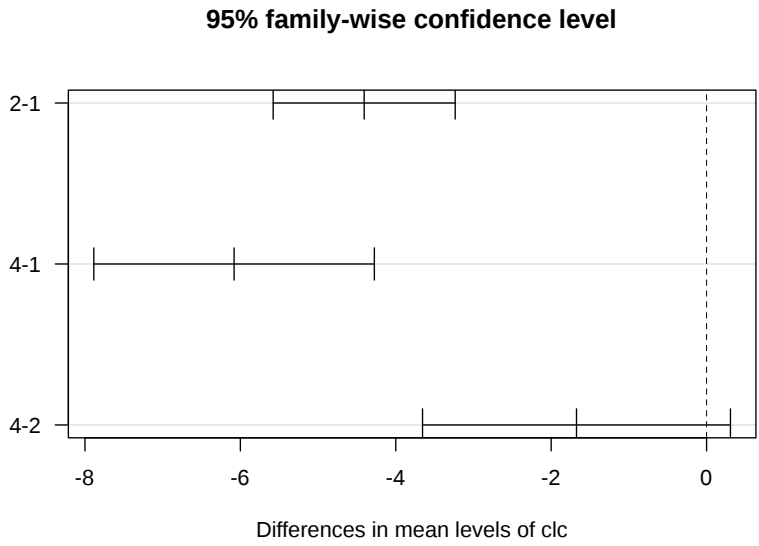
```
tukey = TukeyHSD(aov_test, "clc")
plot(tukey, las = 1)
```

## 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. Przykładowo, możliwe jest stworzenie modelu i estymacji używając logarytmu badanej zmiennej, a następnie przywrócenie oryginalnej jednostki danych.

Redukcja skośności może być przeprowadzona z na wiele sposób. Jednym z najpopularniejszych jest użycie transformacji logarytmicznej.




Rycina 3.10.: Graficzna reprezentacja wyniku porównań wielokrotnych.

### 3.6.2. Testowanie normalności rozkładu

Test Shapiro-Wilka pozwala na sprawdzenie normalności rozkładu wybranej zmiennej.

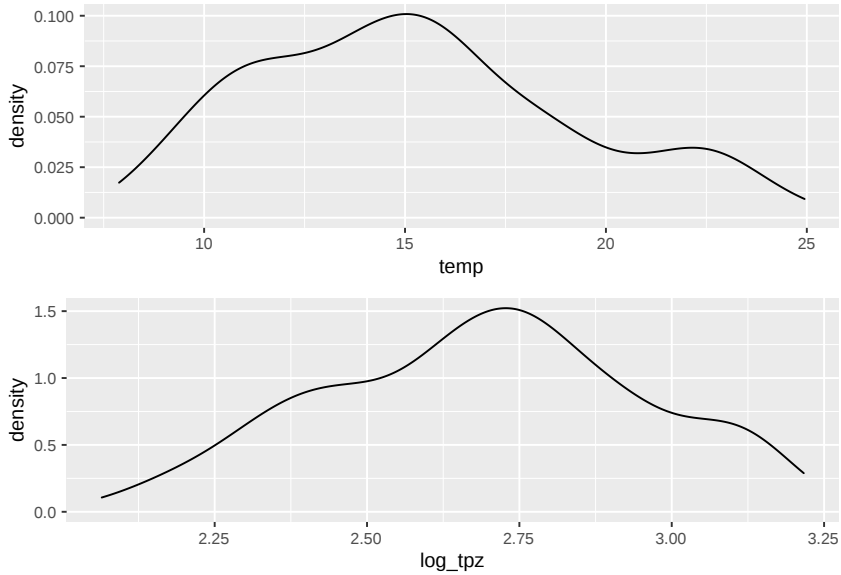
```
shapiro.test(punkty$temp)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  punkty$temp
## W = 0.96904, p-value = 0.00004054
```

### 3.6.3. Logarytmizacja

Jedną grupą technik służących redukcji skośności jest logarytmizacja. Logarytmizacja w R może odbyć się za pomocą funkcji `log()` (rycina 3.11).

```
punkty$log_tpz = log(punkty$temp)
```




Rycina 3.11.: Porównania wartości zmiennej temp przed i po transformacji logaritmicznej.

### 3.6.4. Transformacja odwrotna

Przywrócenie wartości do oryginalnej jednostki, np. po estymacji, wymaga zastosowania odpowiedniej metody transformacji wstecznej (Yamamoto, 2007)[1].

$$y = k_0 \cdot exp[ln(\hat{y}_{OK}) + \frac{\sigma^2_{OK}}{2}]$$

, gdzie:

- $k_0$ - współczynnik korekcyjny (iloraz średniej oryginalnej i średniej po transformacji wstecznej)
- $exp$ - funkcja wykładnicza
- $ln$ - funkcja logarytmiczna
- $\hat{y}_{OK}$ - estymacja krigingu zwykłego
- $\sigma^2_{OK}$ - wariacja krigingu zwykłego

Zobaczmy to na uproszczonym przykładzie. Poniższy blok kodu tworzy model semiwariogramy, a następnie estymację dla badanego obszaru. Wyjaśnienie działania poniższych linii kodu można znaleźć w rozdziałach 6, 7 i 8.

[1] https://link.springer.com/article/10.1007/s10596-007-9046-x

```
library(gstat)
data(siatka)
vario = variogram(log_tpz ~ 1, locations = punkty)
model = fit.variogram(vario, vgm(model = "Sph", nugget = 0.5))
ok = krige(log_tpz ~ 1,
            locations = punkty,
            newdata = siatka,
            model = model)
```

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

Wynik, obiekt `ok` jest rastrem zawierającym dwie warstwy: `var1.pred` i `var1.var`. Pierwsza z nich, `var1.pred`, zawiera estymowane wartości dla każdego oczka siatki, jednak ich jednostkami jest logarytm oryginalnej jednostki ($ln(\hat{y}_{OK})$). Druga warstwa, `var1.var`, opisuje wariancję (niepewność) pomiaru w każdym oczku siatki ($\sigma^2_{OK}$).

Wartości z obu tych warstw możemy podstawić do powyższego wzoru.

```
bt = exp(ok$var1.pred + (ok$var1.var / 2))
```

W kolejnym kroku musimy wyliczyć, $k_0$, współczynnik korekcyjny. Jest to iloraz średniej oryginalnej i średniej po transformacji wstecznej.

```
mu_bt = mean(bt, na.rm = TRUE)
mu_original = mean(punkty$temp, na.rm = TRUE)
k0 = mu_original / mu_bt
```

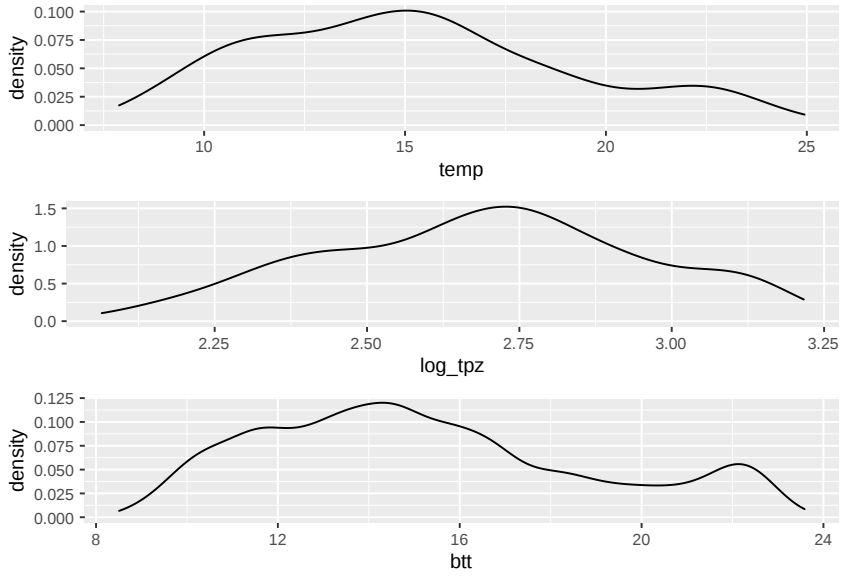
Do otrzymania wynikowych wartości w oryginalnej jednostce musimy teraz pomnożyć wartości po transformacji wstecznej ze współczynnikiem korekcyjnym (rycina 3.12).

```
btt = bt * k0
```

Finalnie możliwe jest też wpisanie wartości po transformacji odwrotnej do siatki.

```
ok$var1.pred2 = btt
```

Aby uprościć sobie transformacje odwrotne w przyszłości, możemy powyższy kod przepisać do postaci funkcji `rev_trans()`:




Rycina 3.12.: Porównania wartości zmiennej temp przed, po transformacji logaritmicznej, oraz po transformacji odwrotnej.

```
rev_trans = function(pred, var, obs){
        bt = exp(pred + (var / 2))
        mu_bt = mean(bt, na.rm = TRUE)
        mu_original = mean(obs, na.rm = TRUE)
        k0 = mu_original / mu_bt
        btt = bt * k0
        return(btt)
}
```

Funkcja `rev_trans()` przyjmuje trzy argumenty:

- `pred` - estymowane wartości, których jednostką jest logarytm oryginalnej jednostki
- `var` - wariancja kolejnych estymacji
- `obs` - wartości pomiarów w punktach w oryginalnej jednostce

Dla ułatwienia funkcja `rev_trans()` znajduje się też w pakiecie **geostatbook**.

## 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 logarytmizację danych na zmiennej `ndvi`. Czy poprawiła ona właściwości rozkładu tej zmiennej?

# 4. Eksploracyjna analiza danych przestrzennych

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

```
library(sf)
library(stars)
library(mapview)
library(geostatbook)
data(punkty)
data(granica)
```

## 4.1. Aspekt przestrzenny

### 4.1.1. Podstawowe terminy

- 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, a następnie wnioskowanie na jej podstawie o populacji.

### 4.1.2. Mapy punktowe

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

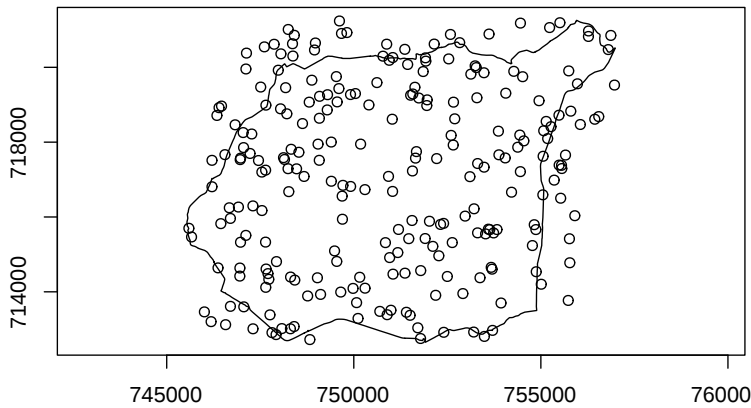
- Sprawdzenie poprawności współrzędnych (sekcja 4.2).
- Sprawdzenie poprawności danych, w tym między innymi określenie danych odstających lokalnie (sekcja 4.3).
- Wgląd w typ próbkowania danych (sekcja 4.4).

- Rozgrupowanie danych przy próbkowaniu preferencyjnym (sekcja 4.5).
- Ogólny pogląd na zmienność przestrzenną, wykorzystanie prostej automatycznej procedury interpolacji (rozdział 5).

## 4.2. Sprawdzenie poprawności współrzędnych

Wstępne sprawdzenie poprawności współrzędnych można wykonać poprzez wizualizację danych przestrzennych za pomocą funkcji `plot()` (rycina 4.1).

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



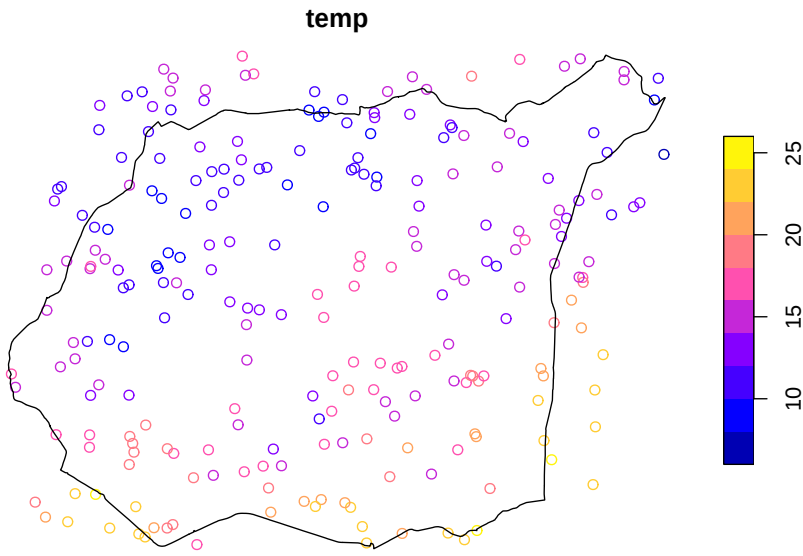

Rycina 4.1.: Lokalizacja punktów pomiarowych na tle granicy obszaru badań.

## 4.3. Dane lokalnie odstające

Dane lokalnie odstające oznaczają nietypowe przestrzennie wartości danej cechy; nie zawsze możliwe jest ich określenie używając statystyk opisowych czy histogramów. Daną lokalnie odstającą może być przykładowo niska wartość otoczona wysokimi wartościami lub też wysoka wartość

otoczona niskimi wartościami. Może to 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 (rycina 4.2).

```
plot(punkty["temp"], reset = FALSE)
plot(st_geometry(granica), add = TRUE)
```




Rycina 4.2.: Wizualizacja zmiennej temp pozwalająca na wyszukanie wartości lokalnie odstających.

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

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

## 4.4. Próbkowanie

### 4.4.1. Podstawowe typy próbowania

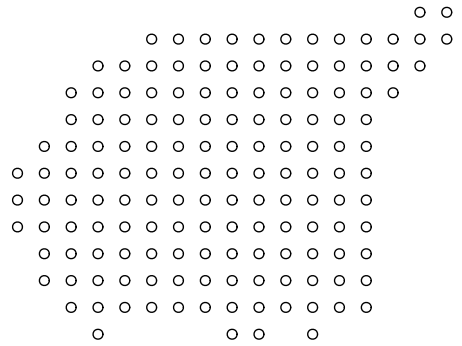
Istnieje cały szereg typów próbkowania danych przestrzennych. Funkcja `st_sample()` z pakietu **sf** pozwala na stworzenie kilku typów próbkowania

(argument `type`), między innymi:

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

### 4.4.2. Regularny typ próbowania

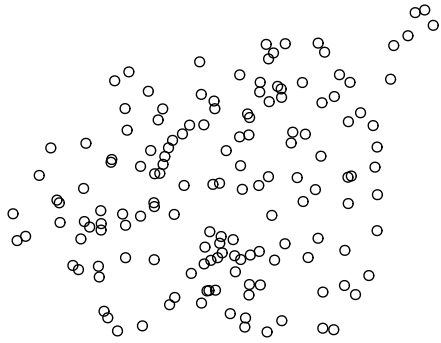
W regularnym typie próbkowania, kolejne punkty rozłożone są równomiernie na badanym obszarze (rycina 4.3).



Rycina 4.3.: Przykład próbkowania regularnego.

### 4.4.3. Losowy typ próbowania

W losowym typie próbkowania każda lokalizacja ma takie samo prawdopodobieństwo wystąpienia. Dodatkowo, każdy punkt jest losowany niezależnie od pozostałych (rycina 4.4).



Rycina 4.4.: Przykład próbkowania losowego.

### 4.4.4. Losowy stratyfikowany typ próbowania

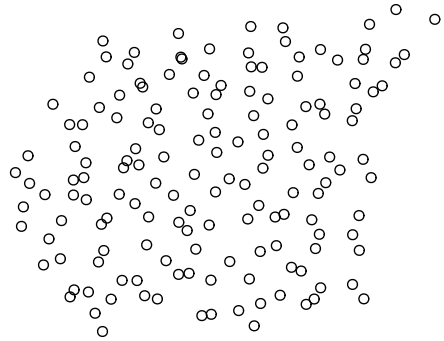
Losowy stratyfikowany typ próbkowania polega na podzieleniu analizowanego obszaru na regularne komórki, a następnie dla każdej komórki losowana jest lokalizacja punktu (rycina 4.5).

### 4.4.5. Preferencyjny typ próbowania

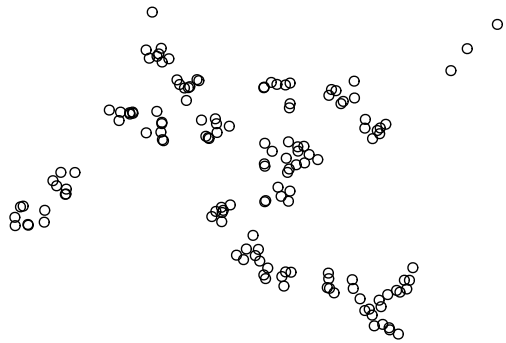
W preferencyjnym typie próbkowania istnieją obszary, które z jakieś powodu (np. specyficzne wartości analizowanej cechy) są znacznie częściej opróbowane niż inne (rycina 4.6).

## 4.5. Rozgrupowanie danych

Typ próbkowania może wpływać na wartości statystyk opisowych. W odpowiedzi na to istenieje szereg metod rozgrupowywania danych, wśród których można wymienić, między innymi, rozgrupowywanie poligonalne oraz rozgrupowywanie komórkowe. Celem tych metod jest nadanie wag obserwacjom w celu zapewnienia reprezentatywności przestrzennej danych.



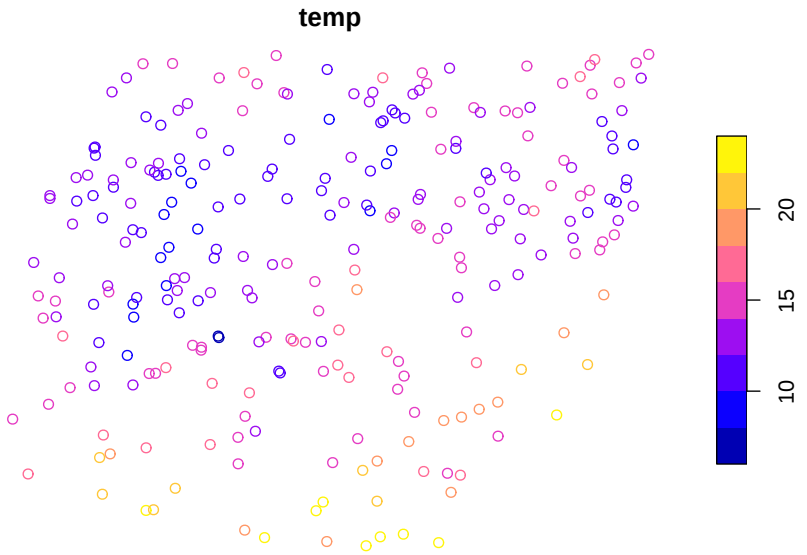
Rycina 4.5.: Przykład próbkowania losowego stratyfikowanego.



Rycina 4.6.: Przykład próbkowania preferencyjnego.

Na potrzeby przykładów związanych z rozgrupowaniem danych w pakiecie **geostatbook** znajduje się zbiór danych `punkty_pref`. W tym zbiorze gęściej opróbkowane są niskie wartości temperatury, co powoduje, że średnia dla całego obszaru jest znacznie niższa niż w rzeczywistości.

```
data(punkty_pref)
plot(punkty_pref["temp"])
```




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

```
## [1] 14.06929
```

### 4.5.1. Rozgrupowanie poligonalne

**Rozgrupowanie poligonalne** polega na zastosowaniu jednej z metod triangulacji, np. poligonów Woronoja:

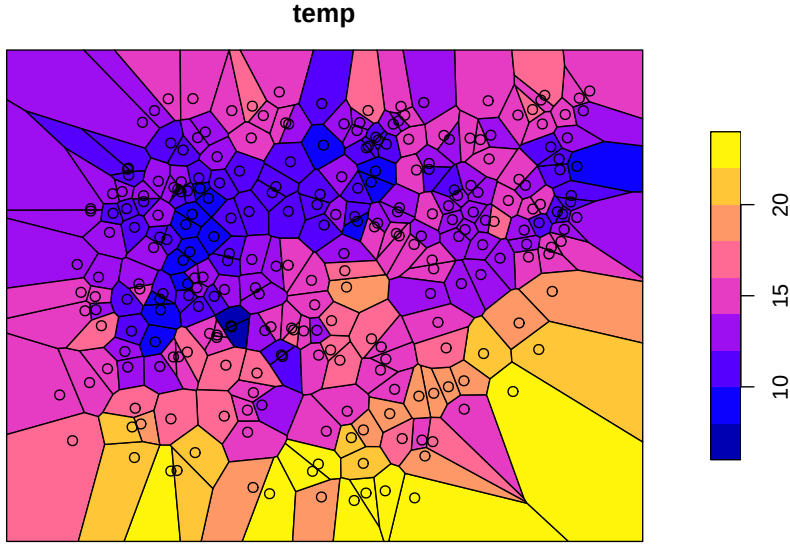
1. Dla każdego punktu określany jest poligon
2. Wyliczana jest powierzchnia poligonu
3. Waga każdego punktu wyliczana jest poprzez podzielenie powierzchni indywidualnych przez powierzchnię całego obszaru, a następnie pomnożenie przez liczbę punktów

$$w'_j = \frac{area_j}{\sum_{j=1}^{n} area_j} \cdot n$$

, gdzie $area_j$ powierzchnia dla wybranej obserwacji, a $n$ to łączna liczba obserwacji

```
library(dismo)
punkty_pref$id = 1:nrow(punkty_pref)
v = st_as_sf(voronoi(as(punkty_pref, "Spatial")))
v$pow = as.numeric(st_area(v))
v$waga_rp = v$pow / sum(v$pow) * nrow(punkty_pref)
v_df = st_drop_geometry(v[c("id", "waga_rp")])
punkty_pref = merge(punkty_pref, v_df, by = "id")
```

```
plot(v["temp"], reset = FALSE)
plot(st_geometry(punkty_pref), add = TRUE)
```




```
srednia_wazona_rp = mean(punkty_pref$temp * punkty_pref$waga_rp, na.rm = TRUE)
srednia_wazona_rp
```

```
## [1] 15.87475
```

### 4.5.2. Rozgrupowanie komórkowe

Rozgrupowanie komórkowe (ang. *cell declustering*) polega na:

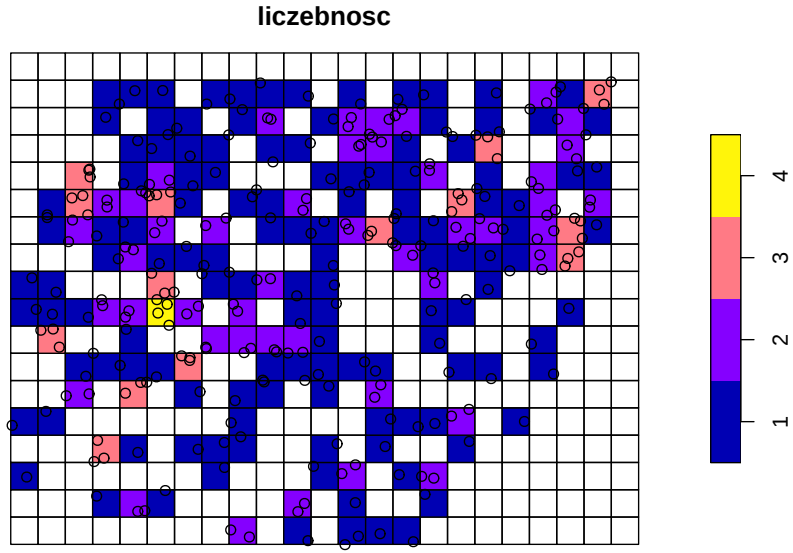
1. Stworzeniu regularnej siatki dla badanego obszaru
2. Policzeniu liczby obserwacji w każdym oczku siatki
3. Nadanie wagi dla każdego punktu, zgodnie ze wzorem:

$$w'_j = \frac{\frac{1}{n_i}}{\text{liczba komorek z danymi}} \cdot n$$

, gdzie $n_i$ to liczba obserwacji w komórce, a $n$ to łączna liczba obserwacji

```
siatka_n = st_make_grid(granica, cellsize = 500)
siatka_n = st_as_sf(siatka_n)
punkty_pref$liczebnosc = 0
st_crs(siatka_n) = st_crs(punkty_pref)
siatka_nr = aggregate(punkty_pref["liczebnosc"], by = siatka_n, FUN = length)
# siatka_nr$odw_licz = 1 / siatka_nr$liczebnosc
punkty_pref$liczebnosc = NULL
punkty_pref = st_join(punkty_pref, siatka_nr)
punkty_pref$waga_rk = ((1 / punkty_pref$liczebnosc) / sum(!is.na(siatka_nr$liczebnosc))) *
        nrow(punkty_pref)
```

```
plot(siatka_nr["liczebnosc"], reset = FALSE)
plot(st_geometry(punkty_pref), add = TRUE)
```




```
srednia_wazona_rk = mean(punkty_pref$temp * punkty_pref$waga_rk, na.rm = TRUE)
srednia_wazona_rk
```

```
## [1] 14.26612
```

### 4.5.3. Porównanie metod rozgrupowania

Średnia wartość temperatury dla badanego obszaru wynosiła 15,59 stopni Celsjusza, jednak w preferencyjnej próbie ta wartość wynosiła 14,07 stopni Celsjusza. Porównując dwie zastosowane metody rozgrupowania warto zauważyć, że najbliższy wynik uzyskano korzystając z rozgrupowania poligonalnego, które nieznacznie zawyżyło rzeczywistą wartość temperatury. Rozgrupowanie komórkowe było w tym przypadku mniej dokładne, wyraźnie zawyżając wartości temperatury.

| | Średnia arytmetyczna |
|---|---:|
| Populacja | 15.5913 |
| Próba | 14.0693 |
| Rozgrupowanie poligonalne | 15.8747 |
| Rozgrupowanie komórkowe | 14.2661 |

W przypadku metod rozgrupowania należy jednak pamiętać, że ich wynik zależy od szeregu wprowadzonych parametrów, w szczególności granic badanego obszaru oraz zastosowanej wielkości oczka siatki.

## 4.6. Zadania

1. Wykonaj mapę pokazującą przestrzenny rozkład wartości NDVI w zbiorze `punkty`. (Dodatkowo: spróbuj użyć do tego pakietu **tmap** - `install.packages("tmap")`).
2. Stwórz nowy obiekt `punkty4326`, który będzie zawierać pomiary z obiektu `punkty`, ale będzie w układzie odwzorowania WGS 84. Wykonaj mapę pokazującą przestrzenny rozkład wartości NDVI w zbiorze `punkty4326`. Czy dostrzegasz jakieś różnice?
3. Określ i opisz typ próbkowania danych `punkty` oraz danych `punkty_pref`.
4. Porównaj średnią arytmetyczną zmiennej `srtm` z obiektu `punkty_pref` ze średnimi uzyskanymi metodami rozgupowania poligonalnego i rozgrupowanie komórkowego. (Dodatkowo: sprawdź różne wielkości oczka siatki w rozgrupowaniu komórkowym).

# 5. Metody interpolacji

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

```
library(sf)
library(stars)
library(gstat)
library(dismo)
library(fields)
library(geostatbook)
data(punkty)
data(siatka)
data(granica)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

Przez przejściem do interpolacji geostatystycznych, którymi są poświęcona kolejne rozdziały, 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 (sekcja 5.2 oraz modele statystyczne (sekcja 5.3).

## 5.1. Tworzenie siatek

Większość metod interpolacji wymaga stworzenia siatki interpolacyjnej (pustego rastra). Istnieją dwa podstawowe rodzaje takich siatek - siatki regularne oraz siatki nieregularne.

### 5.1.1. Siatki regularne

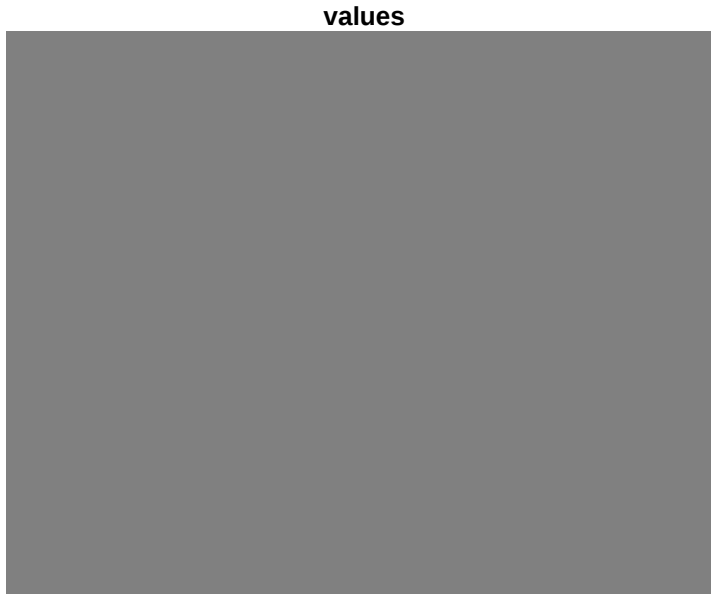
Siatki regularne mają kształt prostokąta obejmującego cały analizowany obszar. Określenie granic obszaru można wykonać na podstawie zasięgu danych punktowych za pomocą funkcji `st_bbox()` z pakietu **sf**.

```
st_bbox(punkty)
```

```
##      xmin      ymin      xmax      ymax
## 745592.5 712723.3 756967.8 721238.7
```

Do stworzenia siatki można wykorzystać funkcję `st_as_stars()`. Tworzy ona nowy obiekt na podstawie obwiedni istniejącego obiektu punktowego oraz zadanej rozdzielczości (rycina 5.1).[1]

```
punkty_bbox = st_bbox(punkty)
nowa_siatka = st_as_stars(punkty_bbox,
                          dx = 500,
                          dy = 500)
nowa_siatka = st_set_crs(nowa_siatka, 2180)
plot(nowa_siatka)
```




Rycina 5.1.: Wizualizacja regularnej siatki.

### 5.1.2. Siatki nieregularne

Siatki nieregularne mają zazwyczaj kształt wieloboku obejmującego analizowany obszar. Mogą one powstać, np. w oparciu o wcześniej istniejące granice.

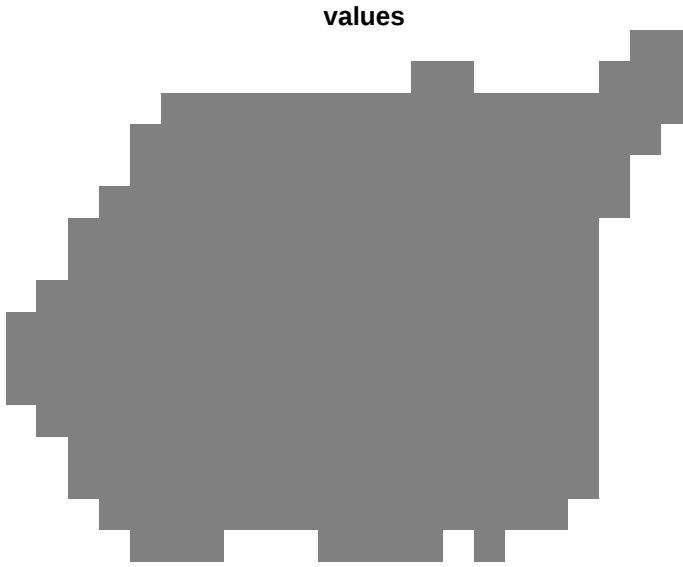
[1] Możliwe jest też ręczne określenie granic obszaru poprzez wpisanie zakresów współrzędnych w funkcji `st_bbox()`.

W poniższym przypadku odczytywana jest granica badanego obszaru z pliku w formacie GeoPackage. Taki obiekt można np. stworzyć za pomocą oprogramowania GIS takiego jak QGIS[2]. Następnie tworzony jest nowy obiekt `nowa_siatka_n` poprzez wybranie tylko tych oczek siatki, które znajdują się wewnątrz zadanych granic.

```
granica = read_sf("dane/granica.gpkg")
nowa_siatka_n = nowa_siatka[granica]
```

Wynik przetworzenia można zobaczyć z użyciem funkcji `plot()` (rycina 5.2).

```
plot(nowa_siatka_n)
```



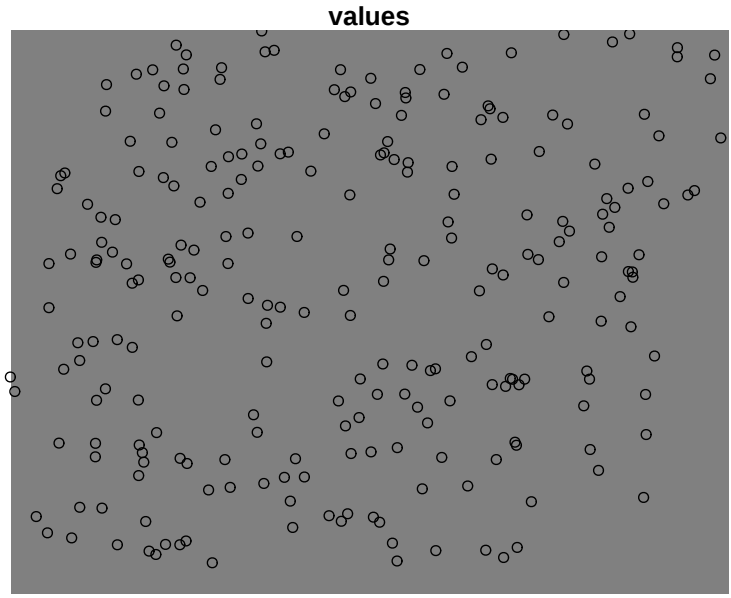

Rycina 5.2.: Wizualizacja nieregularnej siatki.

### 5.1.3. Siatki - wizualizacja

Sprawdzenie, czy uzyskana siatka oraz dane punktowe się na siebie nakładają można sprawdzić za pomocą funkcji `plot()`. W poniższym przykładzie, pierwszy wiersz służy wyświetleniu siatki, a drugi dodaje dane punktowe z użyciem argumentu `add` (rycina 5.3).

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

```
plot(nowa_siatka, reset = FALSE)
plot(st_geometry(punkty), add = TRUE)
```




Rycina 5.3.: Wizualizacja regularnej siatki z nałożonym położeniem obserwacji punktowych.

## 5.2. Modele deterministyczne

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

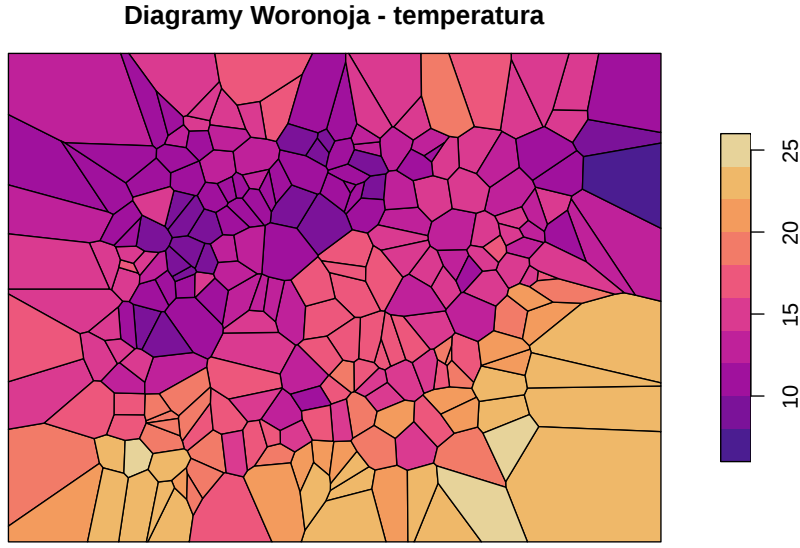
- Metoda diagramów Woronoja (ang. *Voronoi diagram*) (sekcja 5.2.1)
- Metoda średniej ważonej odległością (ang. *Inverse Distance Weighted - IDW*) (sekcja 5.2.2)
- Funkcje wielomianowe (ang. *Polynomials*) (sekcja 5.2.3)
- Funkcje sklejane (ang. *Splines*) (sekcja 5.2.4)

### 5.2.1. Diagramy Woronoja

Metoda diagramów Woronoja polega na stworzeniu nieregularnych poligonów na podstawie analizowanych punktów, a następnie wpisaniu w każdy poligon wartości odpowiadającego mu punktu. Na poniższym przykładzie ta metoda stosowana jest z użyciem funkcji `voronoi()` z pakietu **dismo** (rycina 5.4).

```
voronoi_interp = voronoi(st_coordinates(punkty))
voronoi_interp = st_as_sf(voronoi_interp)
voronoi_interp$temp = punkty$temp
```

```
plot(voronoi_interp["temp"],
     main = "Diagramy Woronoja - temperatura",
     pal = hcl.colors(10, palette = "ag_Sunset"))
```

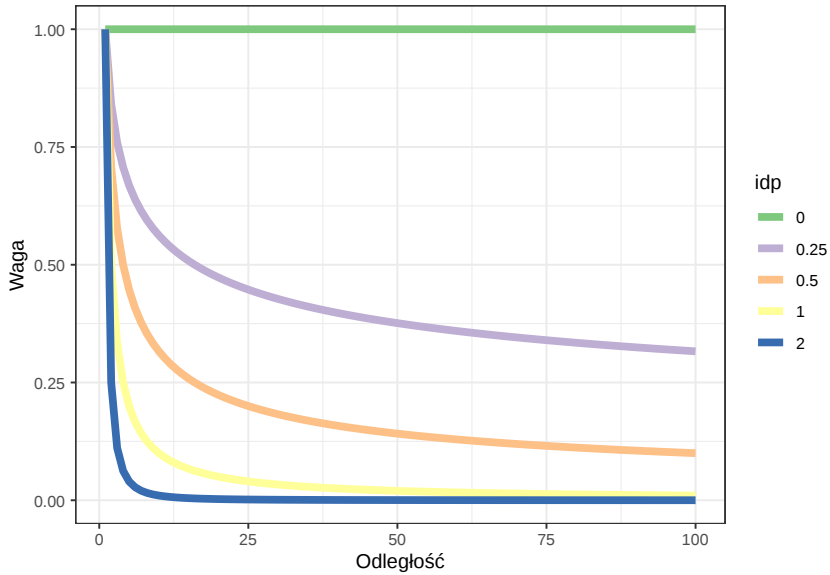



Rycina 5.4.: Interpolacja zmiennej temp używając metody diagramów Woronoja.

### 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 (`idp`, ang. *inverse distance weighting power*) (rycina 5.6).




Rycina 5.5.: Relacja pomiędzy argumentem wykładnika potęgowego a wpływem wartości punktów wraz z odległością.

W pakiecie **gstat** istnieje do tego celu funkcja `idw()`, która przyjmuje analizowaną cechę (`temp~1`), zbiór punktowy, siatkę, oraz wartość wykładnika potęgowego (argument `idp`) (rycina 5.6).

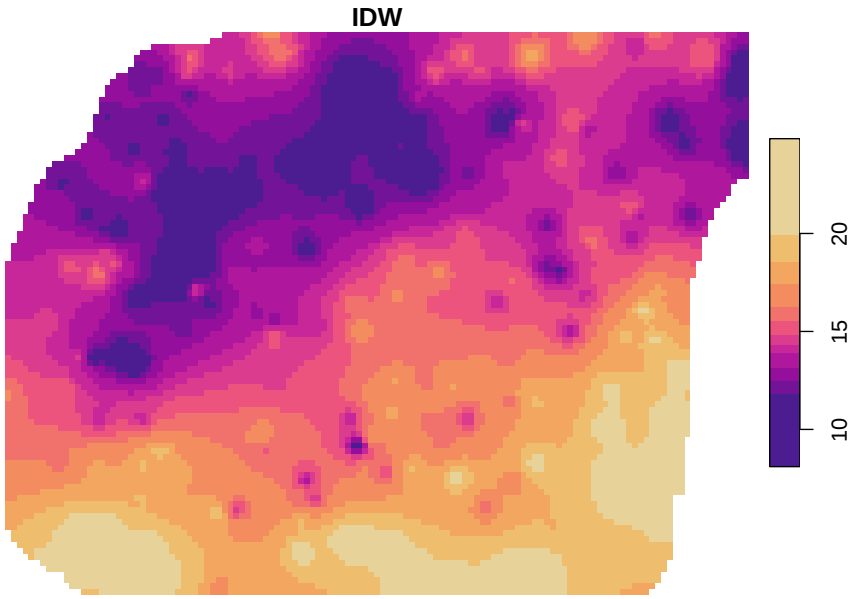
```
idw_interp = idw(temp ~ 1, locations = punkty,
                 newdata = siatka, idp = 2)
```

```
## [inverse distance weighted interpolation]
```

```
plot(idw_interp["var1.pred"], main = "IDW", col = paleta)
```

### 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




Rycina 5.6.: Interpolacja zmiennej temp używając metody średniej ważonej odległością (IDW).

analizowany zbiór danych, oraz `degree` określającą stopień wielomianu. Następnie funkcja `predict()` przenosi nowe wartości na wcześniej stworzoną siatkę. Porównanie wielomianów pierwszego, drugiego i trzeciego stopnia można znaleźć na rycinach 5.7, 5.8 i 5.9.

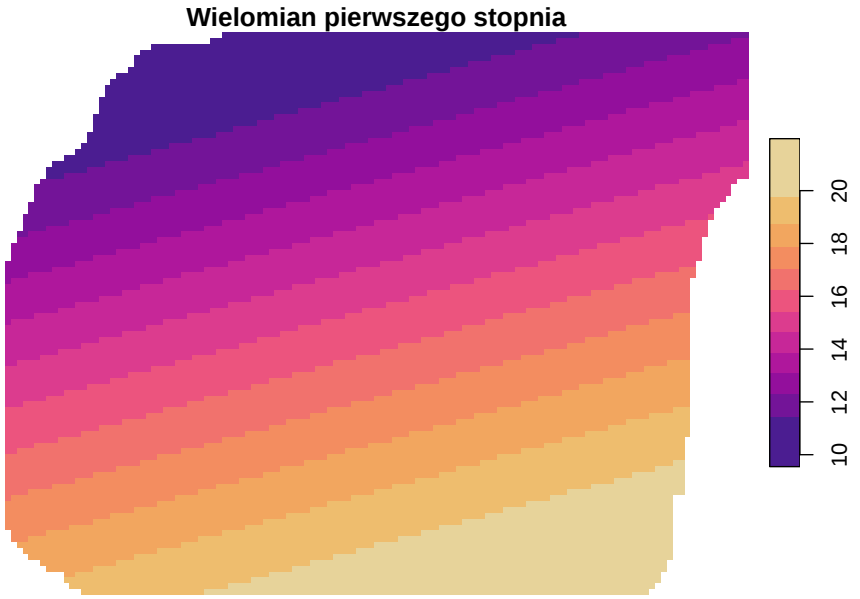
```
# wielomian 1 stopnia
wielomian_1 = gstat(formula = temp ~ 1, locations = punkty,
                    degree = 1)
wielomian_1_pred = predict(wielomian_1, newdata = siatka)
```

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

```
plot(wielomian_1_pred["var1.pred"],
     main = "Wielomian pierwszego stopnia", col = paleta)
```

```
# 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]
```




Rycina 5.7.: Powierzchnia trendu zmiennej temp określona używając wielomianu pierwszego stopnia.

```
plot(wielomian_2_pred["var1.pred"],
     main = "Wielomian drugiego stopnia", col = paleta)
```

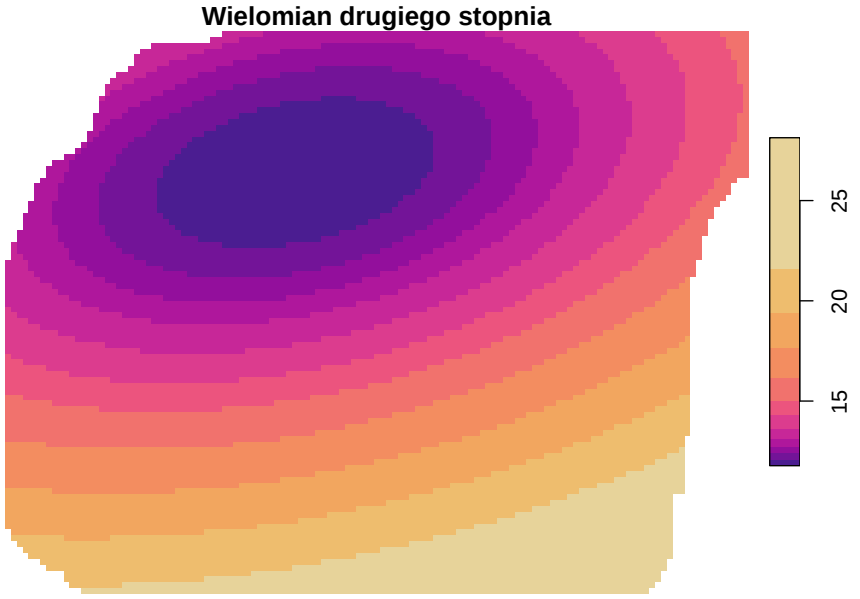
```
# wielomian 3 stopnia
wielomian_3 = gstat(formula = temp ~ 1, locations = punkty,
                    degree = 3)
wielomian_3_pred = predict(wielomian_3, newdata = siatka)
```

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

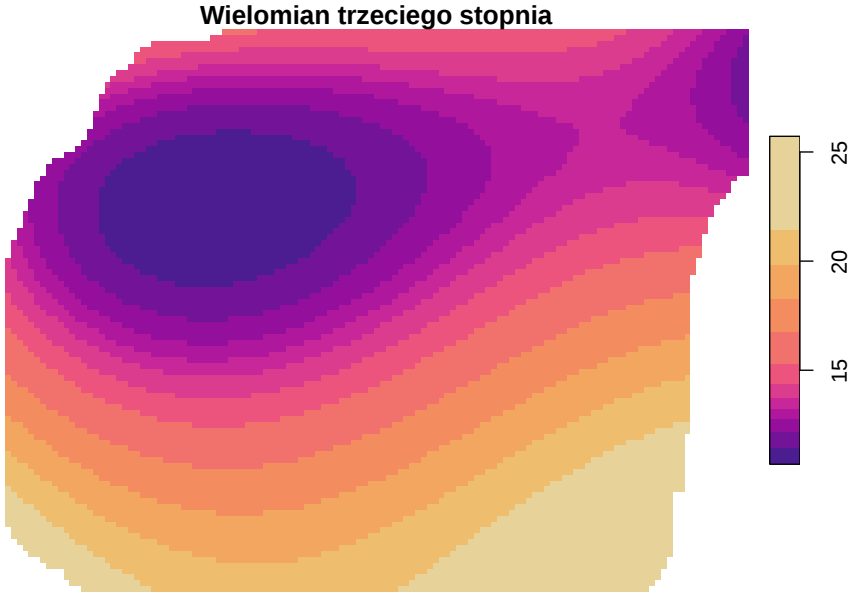
```
plot(wielomian_3_pred["var1.pred"],
     main = "Wielomian trzeciego stopnia", col = paleta)
```

### 5.2.4. Funkcje sklejane

Interpolacja z użyciem funkcji sklejanych (funkcja `Tps()` z pakietu **fields**) dopasowuje krzywą powierzchnię do wartości analizowanych punktów (rycina 5.10).



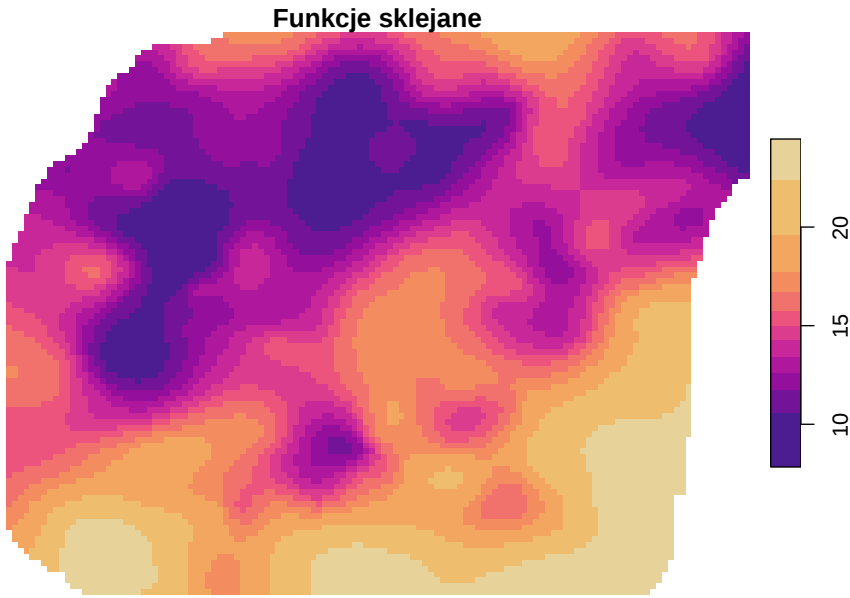

Rycina 5.8.: Powierzchnia trendu zmiennej temp określona używając wielomianu drugiego stopnia.




Rycina 5.9.: Powierzchnia trendu zmiennej temp określona używając wielomianu trzeciego stopnia.

```
tps = Tps(st_coordinates(punkty), punkty$temp)
siatka$tps_pred = predict(tps, st_coordinates(siatka))
siatka$tps_pred[is.na(siatka$X2)] = NA
```

```
plot(siatka["tps_pred"],
     main = "Funkcje sklejane", col = paleta)
```



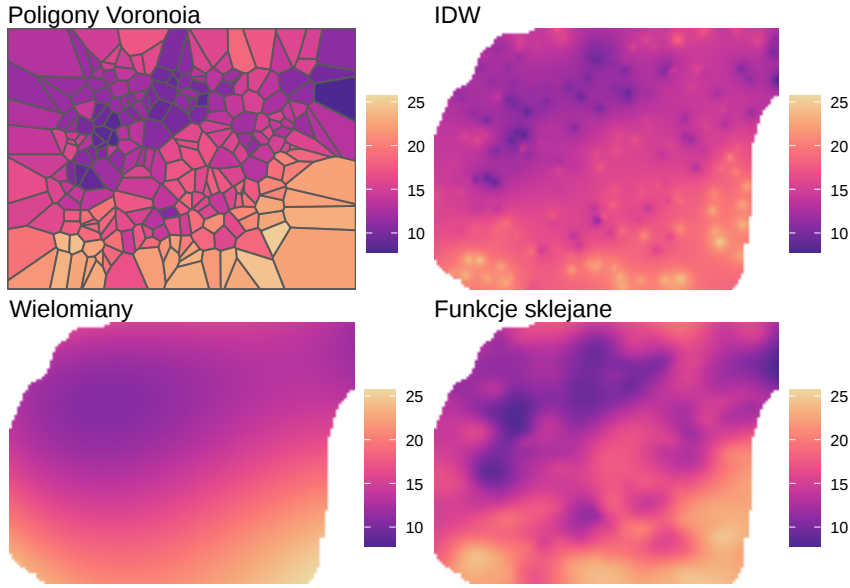

Rycina 5.10.: Interpolacja zmiennej temp z użyciem funkcji sklejanych.

### 5.2.5. Porównanie modeli deterministycznych

Na rycinie 5.11 można wizualnie porównać wyniki uzyskane czterema metodami deterministycznymi.

## 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. Te metody jednak zazwyczaj wymagają większych zasobów sprzętowych. Do modeli statystycznych należą, między innymi:




Rycina 5.11.: Porównanie czterech metod interpolacji dla zmiennej temp używając jednolitej skali wartości.

- Kriging
- Modele regresyjne
- Modele bayesowskie
- Modele hybrydowe

W kolejnych rozdziałach znajduje się 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(sf)
library(stars)
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 geostatystyki

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:

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

### 6.1.5. Badane zjawisko

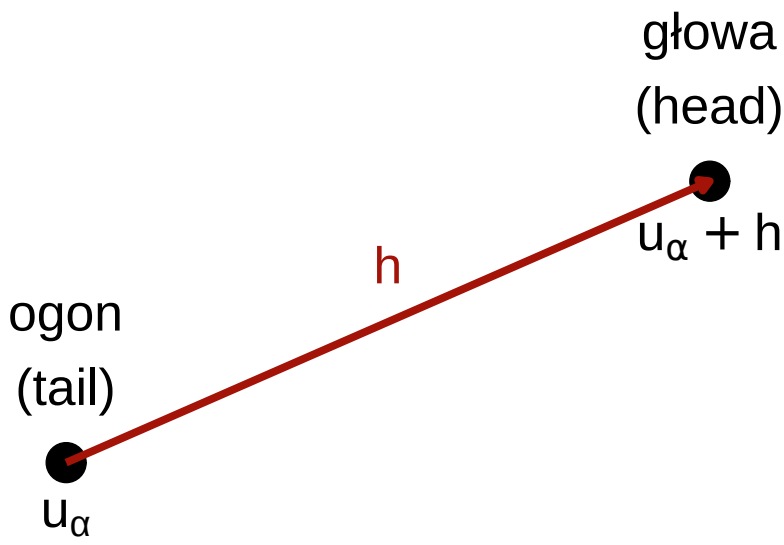
Oprócz ograniczeń dotyczących danych wejściowych, istnieją również dwa założenia dotyczące analizowanej cechy (analizowanego zjawiska):

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 (rycina 6.1) to:

- $u_\alpha$ - wektor współrzędnych.
- $z(u_\alpha)$ - badana zmienna jako funkcja położenia - inaczej określany jako ogon (ang. *tail*).
- $h$ - lag - odstęp pomiędzy dwoma lokalizacjami.
- $z(u_\alpha + h)$ - wartość badanej zmiennej odległej o odstęp $h$ - inaczej określany jako głowa (ang. *head*).




Rycina 6.1.: Wizualizacja relacji pomiędzy wartością ogona a głowy odległymi od siebie o wektor h.

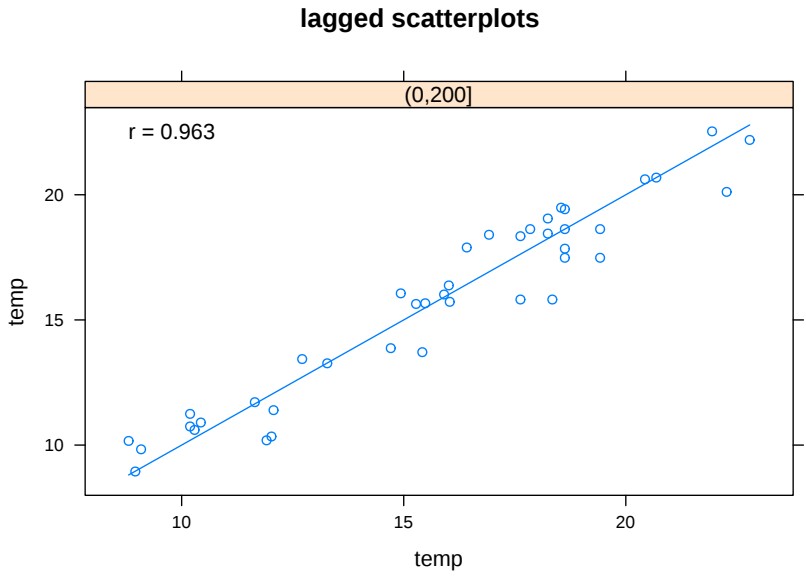
## 6.2. Przestrzenna kowariancja i korelacja

### 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 (rycina 6.2).



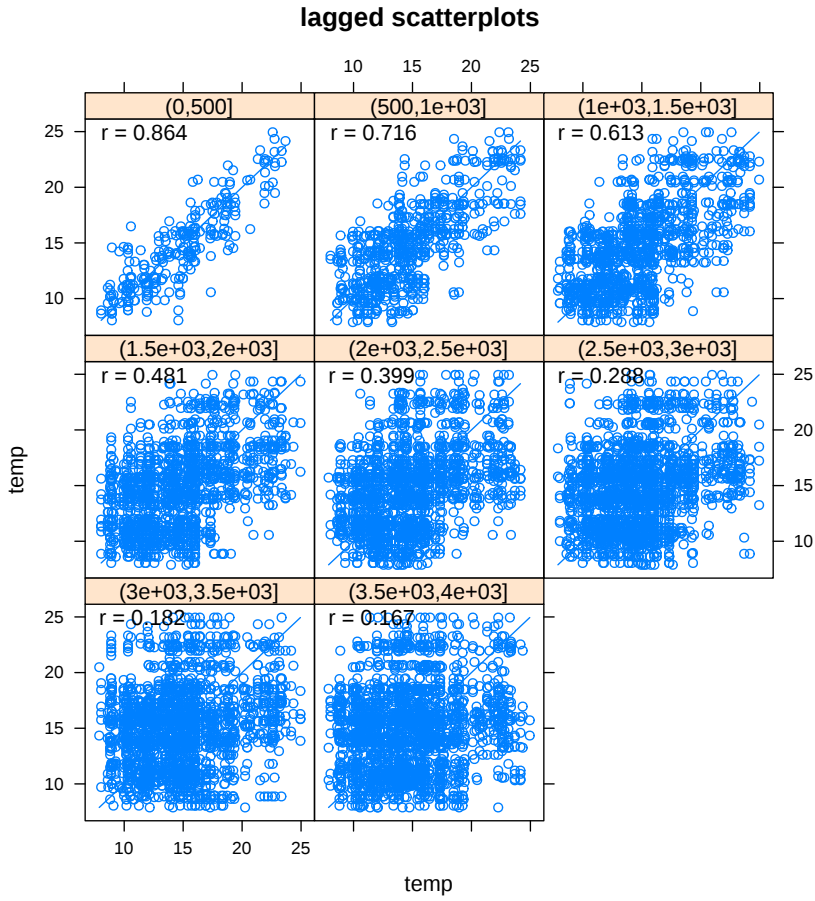

Rycina 6.2.: Relacja między wartościami zmiennej temp w danym miejscu a wartościami zmiennej temp w odległości do 200 metrów.

## 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** (rycina 6.3).

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

Przykładowo, na powyższym 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.

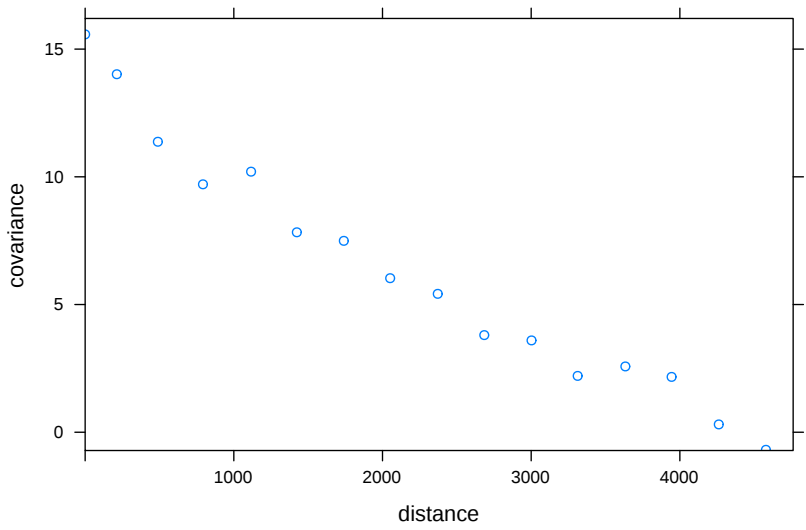



Rycina 6.3.: Wykres rozrzutu z przesunięciem dla zamiennej temp.

### 6.2.3. Autokowariancja

Podobną informację jak wykres rozrzutu z przesunięciem daje autokowariancja (rycina 6.4). 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)
```

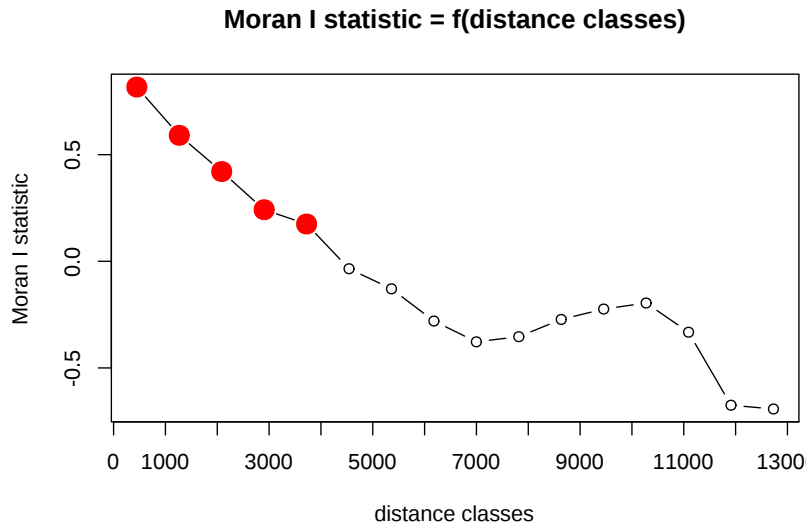



Rycina 6.4.: Autokowariancja zmiennej temp.

### 6.2.4. Autokorelacja

Kolejną miarą przestrzennego podobieństwa jest autokorelacja (rycina 6.5). 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 = st_coordinates(punkty)
kor = correlog(wsp, punkty$temp, method = "Moran")
plot(kor)
```




Rycina 6.5.: Autokorelacja zmiennej temp.

## 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. Jednocześnie warto zdawać sobie sprawę, że kowarancja i korelacja przestrzenna nadaje się nie gorzej niż semiwariancja dla potrzeb interpretacji relacji przestrzennych.

### 6.3.2. Wzór

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

$$\gamma(h) = \frac{1}{2}(z(u_\alpha) - z(u_\alpha + h))^2$$

Przykładowo, aby wyliczyć wartość semiwariancji (`gamma`) pomiędzy dwoma punktami musimy znać wartość pierwszego z nich (w przykładzie jest to ok. 13,85 stopni Celsjusza) oraz drugiego z nich (ok. 15,48 stopni Celsjusza). Korzystając z wzoru na semiwariację otrzymujemy wartość równą ok. 1,33. Znamy również odległość między punktami (ok. 3240,89 metra), więc możemy w uproszczeniu stwierdzić, że dla tej pary punktów odległych o 3240 metrów wartość semiwariancji wynosi około 1,33.

```
odl = st_distance(punkty[c(1, 2), ])[2]
gamma = 0.5 * (punkty$temp[1] - punkty$temp[2]) ^ 2
gamma
```

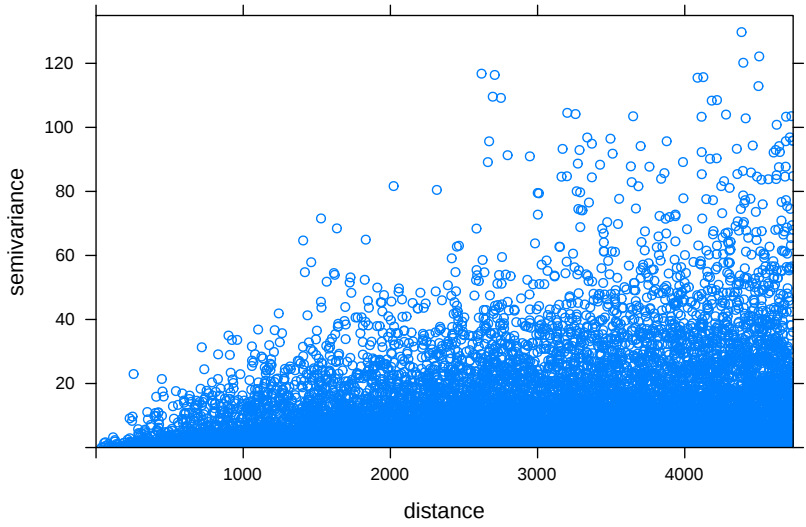
```
## [1] 1.331691
```

### 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 (rycina 6.6). 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 (rycina 6.7). Po skończonym wyborze należy nacisnąć klawisz Esc.




Rycina 6.6.: Chmura semiwariogramu zmiennej temp.

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

Następnie można wyświetlić wybrane pary punktów z użyciem funkcji `plot()` (rycina 6.8).

```
plot(vario_cloud_sel, punkty)
```

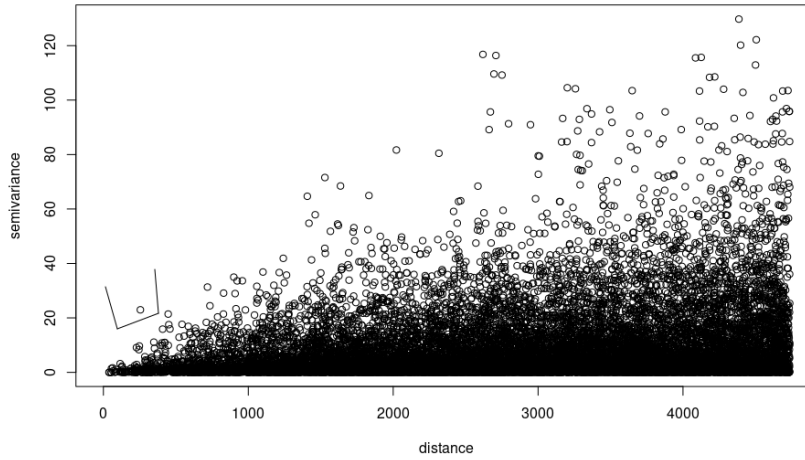
### 6.3.4. Charakterystyka struktury przestrzennej

Semiwariogram jest wykresem pokazującym relację pomiędzy odległością a semiwariancją (rycina 6.9). Inaczej mówiąc, dla kolejnych odstępów (lagów) wartość semiwariancji jest uśredniana i przestawiania w odniesieniu do odległości.

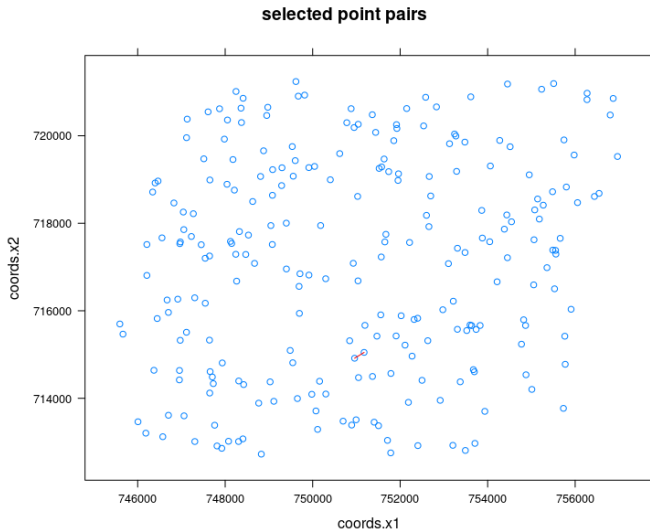
$$\hat{\gamma}(h) = \frac{1}{2N(h)} \sum_{\alpha=1}^{N(h)} (z(u_\alpha) - z(u_\alpha + h))^2$$

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

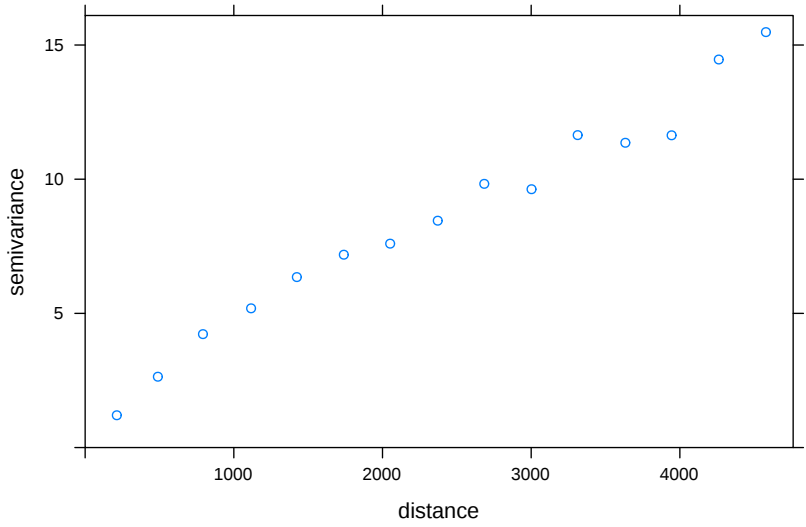
W oparciu o semiwariogram empiryczny (czyli oparty na danych) możemy następnie dopasować do niego model/e.




Rycina 6.7.: Przykład wyboru par odstających z chmury semiwariogramu zmiennej temp.




Rycina 6.8.: Lokalizacja wybranych par obserwacji (czerwone linie).




Rycina 6.9.: Semiwariogram empiryczny zmiennej temp.

### 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`).

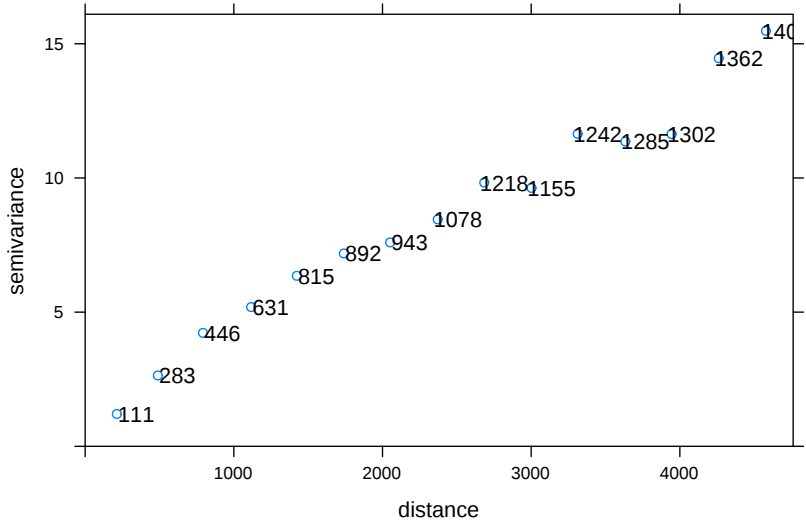
```
vario_par = variogram(temp ~ 1, locations = punkty)
vario_par
```

```
##       np      dist     gamma dir.hor dir.ver   id
## 1    111  212.9356  1.203287       0       0 var1
## 2    283  489.0804  2.639854       0       0 var1
## 3    446  792.3222  4.226593       0       0 var1
## 4    631 1116.0957  5.187903       0       0 var1
## 5    815 1423.8256  6.352106       0       0 var1
## 6    892 1739.7600  7.186529       0       0 var1
## 7    943 2051.5149  7.598215       0       0 var1
## 8   1078 2371.6198  8.453746       0       0 var1
## 9   1218 2684.4065  9.826903       0       0 var1
## 10 1155 3002.3276  9.626642       0       0 var1
## 11 1242 3312.6962 11.642621       0       0 var1
## 12 1285 3633.4536 11.357612       0       0 var1
## 13 1302 3944.5241 11.635848       0       0 var1
```

```
## 14 1362 4261.7974 14.460067       0       0 var1
## 15 1409 4578.9006 15.480848       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` (rycina 6.10).

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




Rycina 6.10.: Semiwariogram empiryczny zmiennej temp wraz z zaznaczoną liczbą par dla każdego odstępu.

### 6.3.6. Parametry semiwariogramu

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

W zrozumieniu danych oraz przy określaniu parametrów semiwariogramu może pomóc szereg obliczeń pomocniczych. Przykładowo, aby wyliczyć liczbę par obserwacji można użyć poniższego kodu.

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

```
## [1] 29161
```

Powierzchnia zajmowana przez jedną próbkę jest osiągana poprzez podzielenie całej badanej powierzchni poprzez liczbę punktów.

```
pow_pr = st_area(granica) / nrow(punkty)
pow_pr
```

```
## 261886.9 [m^2]
```

Średnia odległość między punktami to wartość pierwiastka z powierzchni zajmowanej przez jedną próbkę.

```
sqrt(pow_pr)
```

```
## 511.7489 [m]
```

Ostatnim obliczeniem pomocniczym jest określenie połowy pierwiastka powierzchni. Może być ono następnie użyte do ustalenia maksymalnego zasięgu semiwariogramu.

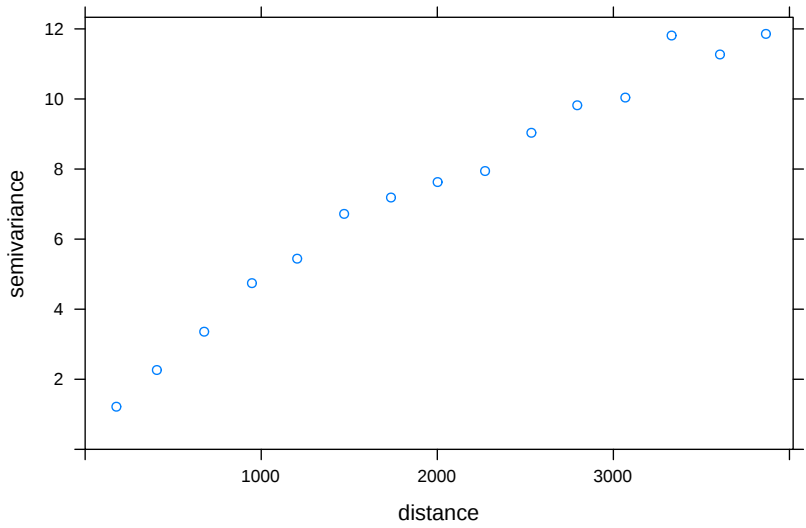
```
pow = st_area(granica)
0.5 * sqrt(pow)
```

```
## 3980.472 [m]
```

### 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` (rycina 6.11).

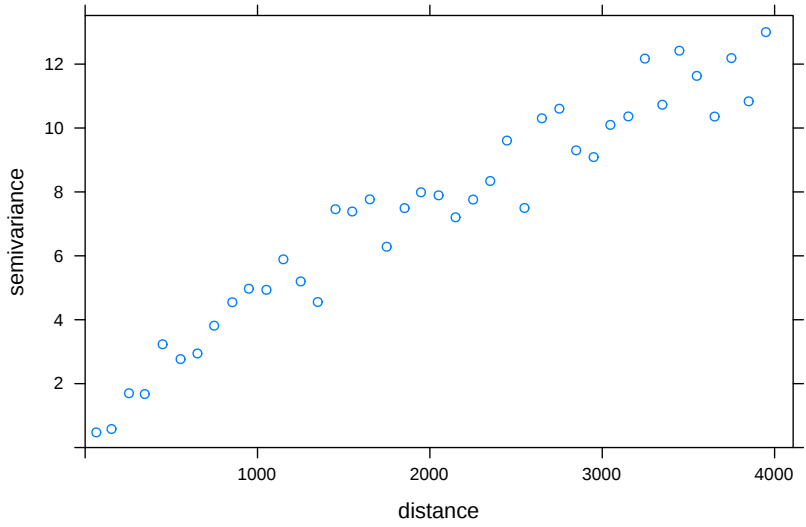
```
vario_par = variogram(temp ~ 1, locations = punkty,
                      cutoff = 4000)
plot(vario_par)
```




Rycina 6.11.: Semiwariogram empiryczny zmiennej temp używając ręcznie ustalonej wartości maksymalnego zasięgu semiwariogramu.

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 (rycina 6.12).

```
vario_par = variogram(temp ~ 1, locations = punkty,
                      cutoff = 4000, width = 100)
plot(vario_par)
```




Rycina 6.12.: Semiwariogram empiryczny zmiennej temp używając ręcznie ustalonej wartości szerokości odstępów.

## 6.4. Anizotropia

### 6.4.1. Anizotropia struktury przestrzennej

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

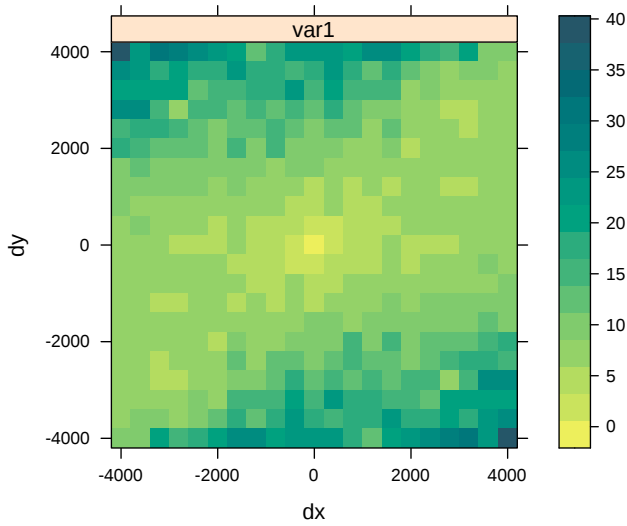
### 6.4.2. Mapa semiwariogramu

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()` (rycina 6.13). 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)
# co najmniej 30 par punktów
plot(vario_map, threshold = 30,
     col.regions = hcl.colors(40, palette = "ag_GrnYl", rev = TRUE))
```

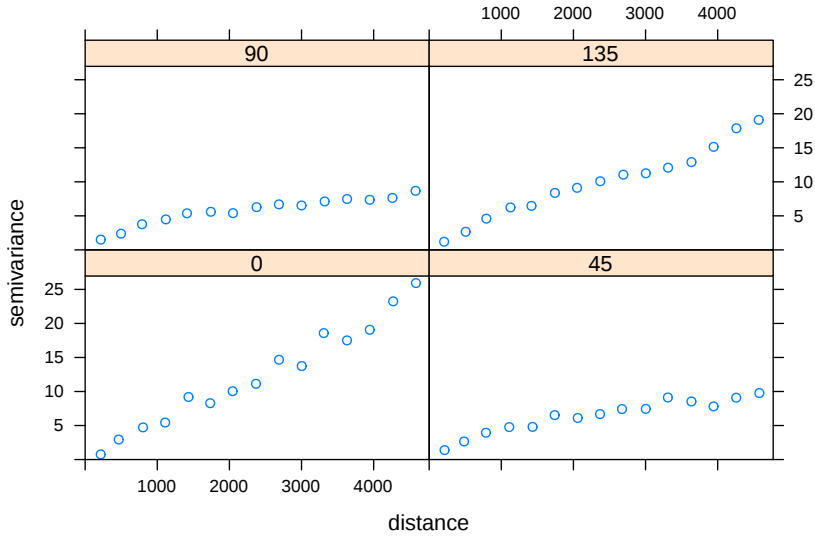



Rycina 6.13.: Mapa semiwariogramu zmiennej temp.

### 6.4.3. Semiwariogramy kierunkowe

W przypadku, gdy zjawisko wykazuje anizotropię przestrzenną, możliwe jest stworzenie semiwariogramów dla różnych wybranych kierunków (rycina 6.14). 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)
```




Rycina 6.14.: Semiwariogramy kierunkowe dla zmiennej temp.

## 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(sf)
library(stars)
library(gstat)
library(geostatbook)
data(punkty)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 7.1. Modelowanie matematycznie autokorelacji przestrzennej

### 7.1.1. Modelowanie struktury przestrzennej

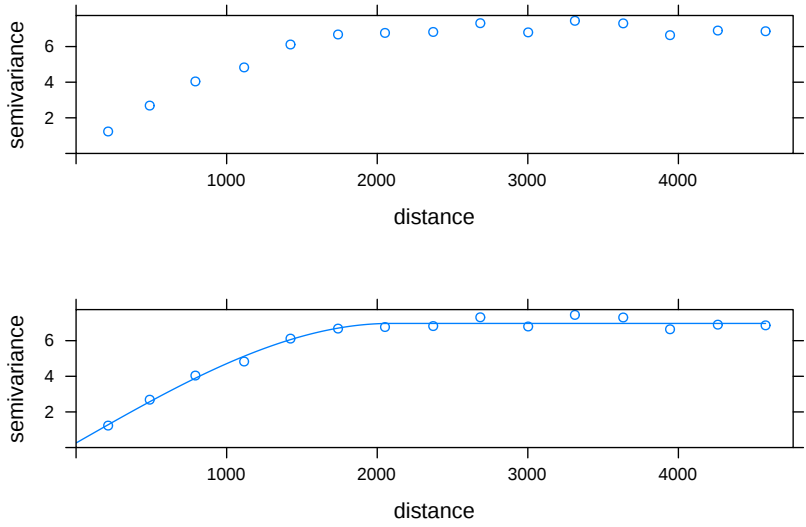
Semiwariogram empiryczny (wyliczony z danych punktowych) jest:

- 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 (rycina 7.1).

### 7.1.2. Model semiwariogramu

Model semiwariogramu składa się zazwyczaj z trzech podstawowych elementów (rycina 7.2). Są to:

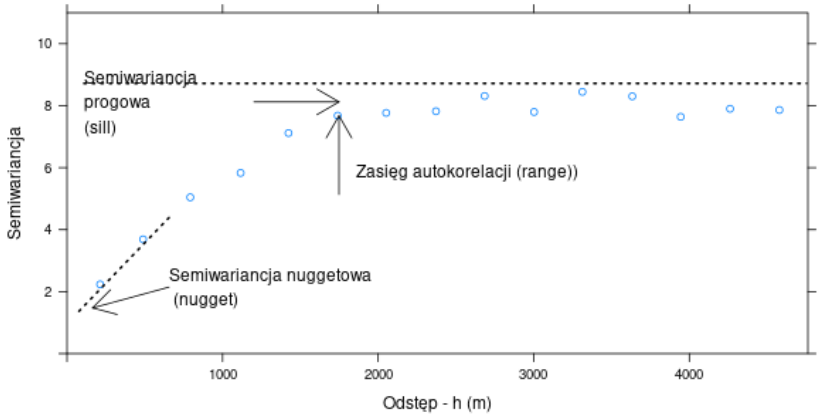



Rycina 7.1.: Porównanie semiwariogramu empirycznego i modelu semiwariogramu.

- **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.




Rycina 7.2.: Podstawowe elementy modelu semiwariogramu.

## 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()`.

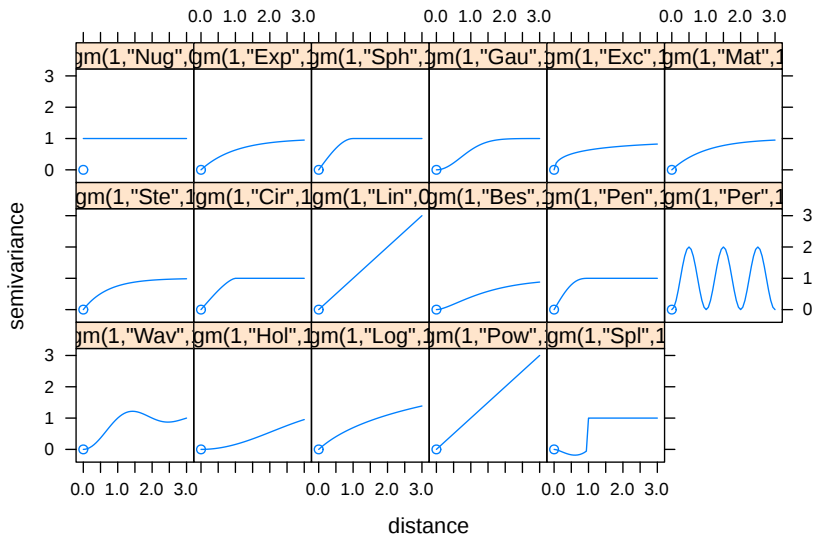
```
vgm()
```

```
##    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()` (rycina 7.3).

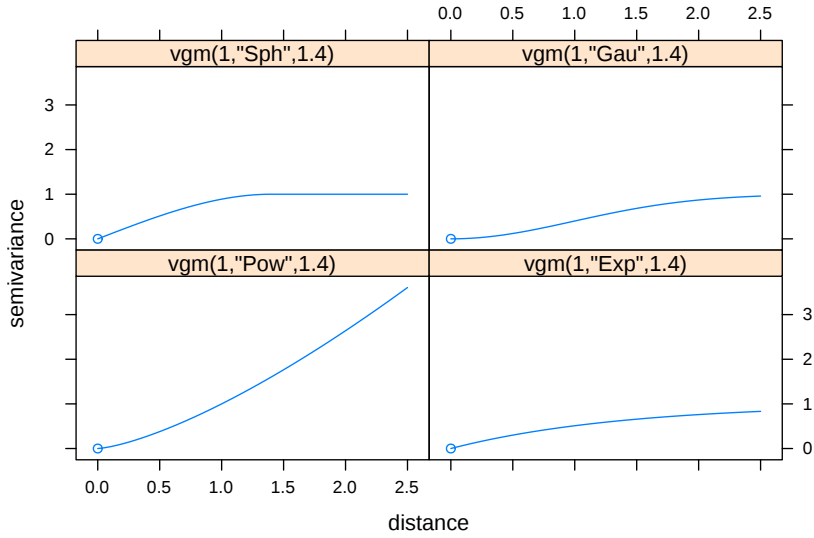
```
show.vgms()
```




Rycina 7.3.: Przykład modeli semiwariogramu dostępnych w pakiecie gstat.

Istnieje możliwość wyświetlenia tylko wybranych modeli podstawowych poprzez argument `models` (rycina 7.4).

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




Rycina 7.4.: Porównanie modeli sferycznego (Sph), gaussowskiego (Gau), potęgowego (Pow) i wykładniczego (Exp).

Dodatkowo, można je porównać na jednym wykresie poprzez argument `as.groups = TRUE` (rycina 7.5).

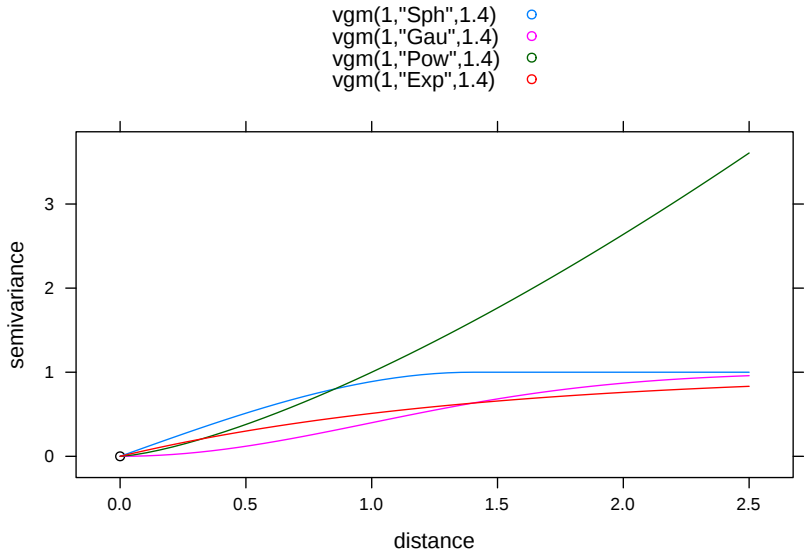
```
show.vgms(models = c("Sph", "Gau", "Pow", "Exp"),
          range = 1.4, max = 2.5, as.groups = TRUE)
```

## 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**.




Rycina 7.5.: Porównanie modeli sferycznego (Sph), gaussowskiego (Gau), potęgowego (Pow) i wykładniczego (Exp) na jednym wykresie.

- 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[1].

[1] Więcej na temat działania tej funkcji można przeczytać we wpisie na stronie https://www.r-spatial.org/r/2016/02/14/gstat-variogram-fitting.html.

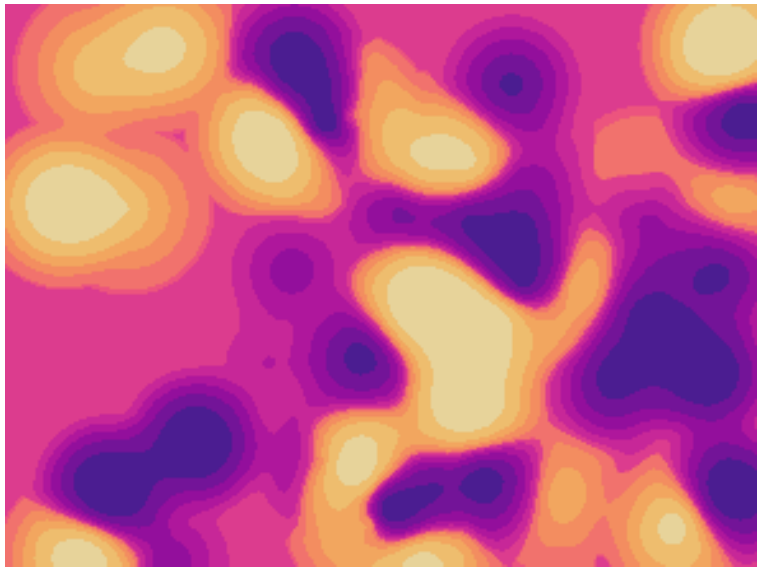
## 7.4. Modelowanie izotropowe

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

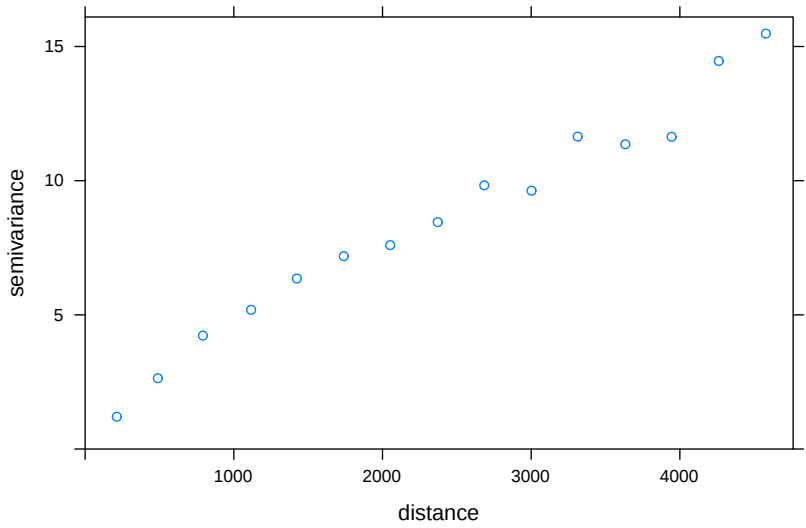
Model sferyczny (`Sph`) jest jednym z najczęściej stosowanych modeli geostatystycznych. Reprezentuje on cechę, której zmienność wartości ma charakter naprzemiennych płatów niskich i wysokich wartości (rycina 7.6). Średnio te płaty mają średnicę określoną przez zasięg (`range`) modelu.



Rycina 7.6.: Przykład zjawiska reprezentowanego poprzez model sferyczny.

Stworzenie takiego modelu dla zmiennej `temp` polega najpierw na zbudowaniu semiwariogramu empirycznego używając funkcji `variogram()` (rycina 7.7).

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




Rycina 7.7.: Semiwariogram zmiennej temp.

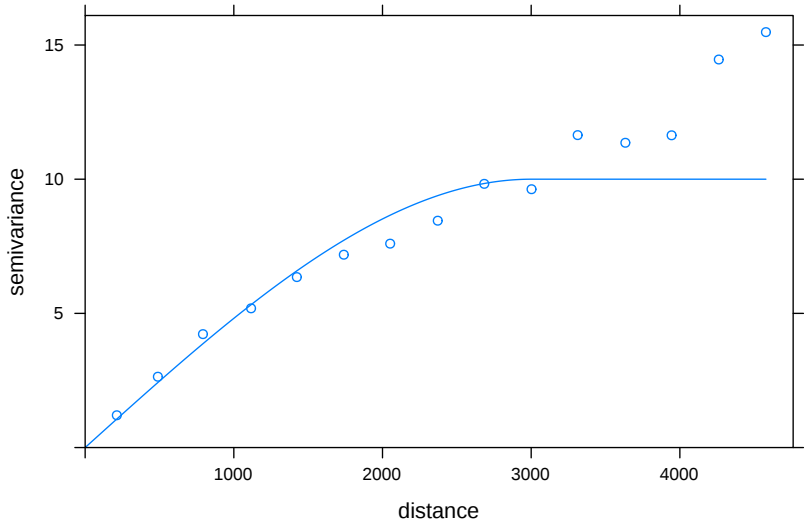
W kolejnym kroku, przy użyciu funkcji `vgm()` określa się typ modelu oraz jego podstawowe parametry (rycina 7.8).

```
model_sph = vgm(psill = 10, model = "Sph", range = 3000)
model_sph
```

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

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

Dodatkowo możliwe jest także automatyczne dopasowanie modelu w oparciu o wstępnie podane parametry używając funkcji `fit.variogram()` (rycina 7.9).




Rycina 7.8.: Model sferyczny zmiennej temp.

```
fitted_sph = fit.variogram(vario, model_sph)
fitted_sph
```

```
##   model    psill    range
## 1   Sph 13.20295 4549.691
```

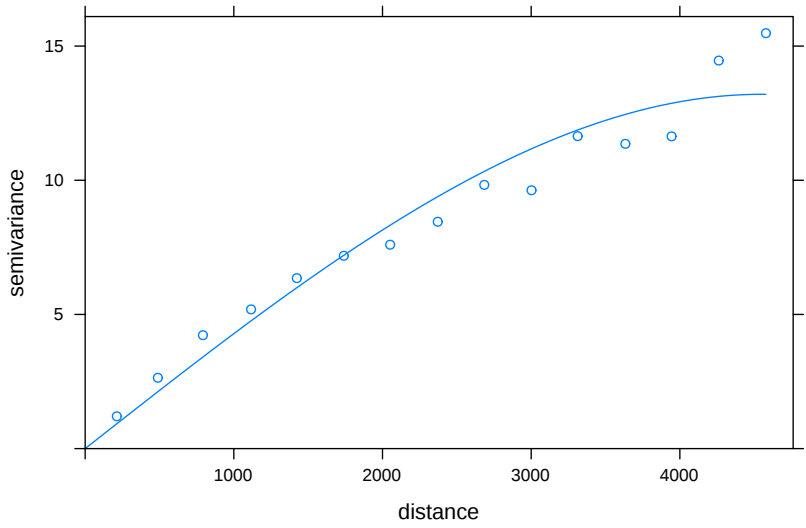
```
plot(vario, model = fitted_sph)
```

W niektórych przypadkach funkcja `fit.variogram()` da także zadawalające wyniki, gdy tylko zostanie podany typ modelu (rycina 7.10).

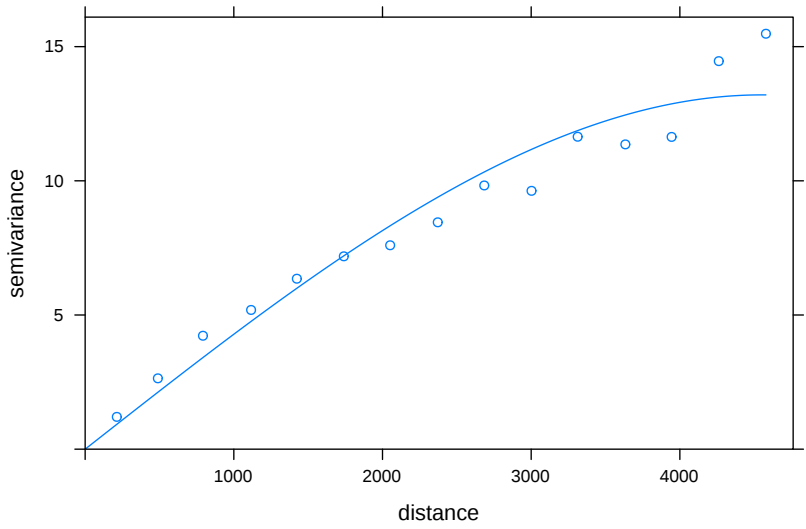
```
model_sph2 = vgm(model = "Sph")
fitted_sph2 = fit.variogram(vario, model_sph2)
fitted_sph2
```

```
##   model   psill    range
## 1   Sph 13.2018 4549.092
```

```
plot(vario, model = fitted_sph2)
```



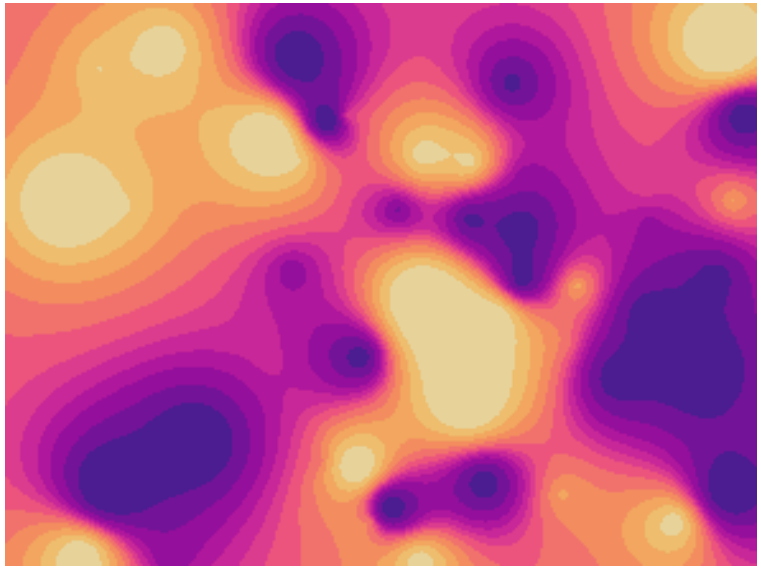

Rycina 7.9.: Automatycznie dopasowany model sferyczny zmiennej temp używając wstępnie podanych parametrów.




Rycina 7.10.: Automatycznie dopasowany model sferyczny zmiennej temp używając jedynie wstępnie podanego typu modelu.

### 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 (rycina 7.11).



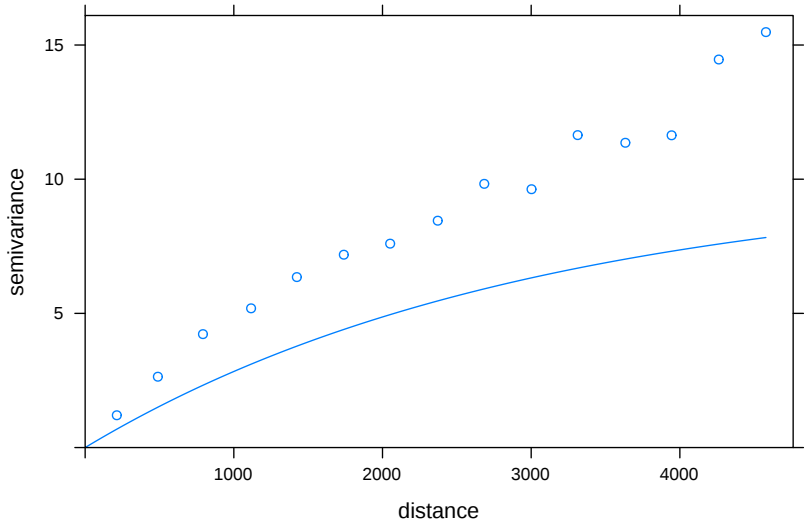
Rycina 7.11.: Przykład zjawiska reprezentowanego poprzez model wykładniczy.

Proces tworzenia tego modelu jest bardzo podobny do przedstawionego powyżej - budowany jest semiwariogram empiryczny, a następnie ustalany jest model poprzez podanie kilku jego parametrów (rycina 7.12).

```
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)
```




Rycina 7.12.: Model wykładniczy zmiennej temp.

Dalej, funkcja `fit.variogram()` może pomóc w dopasowaniu tego modelu (rycina 7.13).

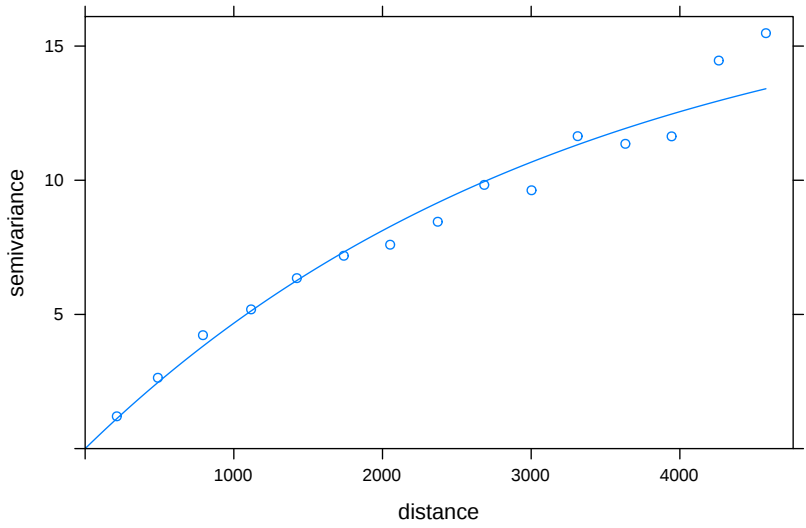
```
fitted_exp = fit.variogram(vario, model_exp)
fitted_exp
```

```
##   model    psill    range
## 1   Exp 17.87051 3298.917
```

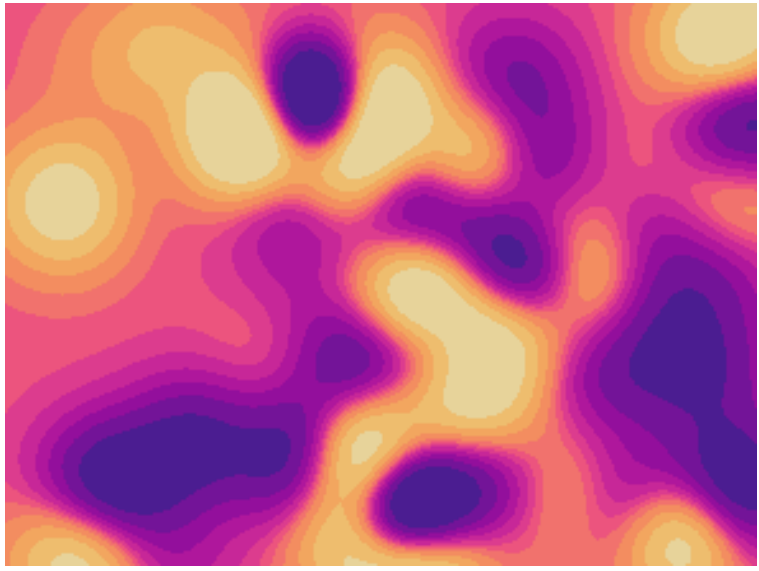
```
plot(vario, model = fitted_exp)
```

### 7.4.3. Model gaussowski

Model gaussowski (`Gau`) również posiada zasięg praktyczny definiowany jako 95% wartości wariancji progowej (rycina 7.14). 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.




Rycina 7.13.: Automatycznie dopasowany model wykładniczy zmiennej temp.



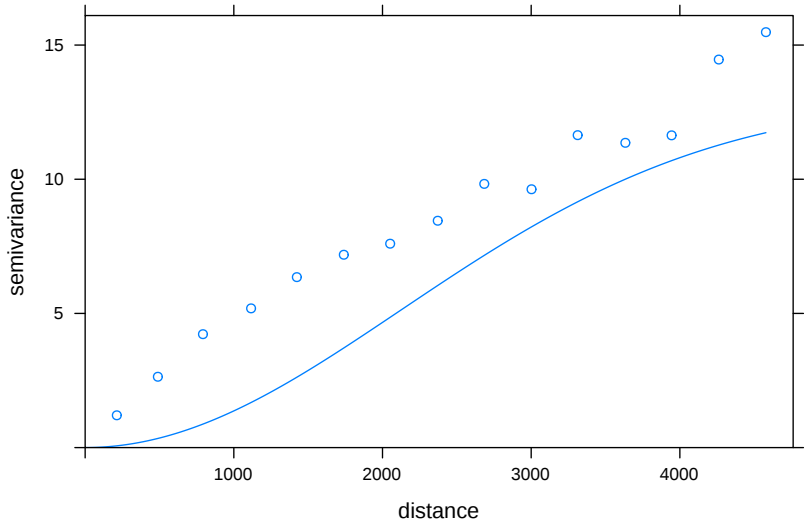
Rycina 7.14.: Przykład zjawiska reprezentowanego poprzez model gaussowski.

Zdefiniowanie modelu gaussowskiego odbywa się używając skrótu `"Gau"` (rycina 7.15).

```
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)
```



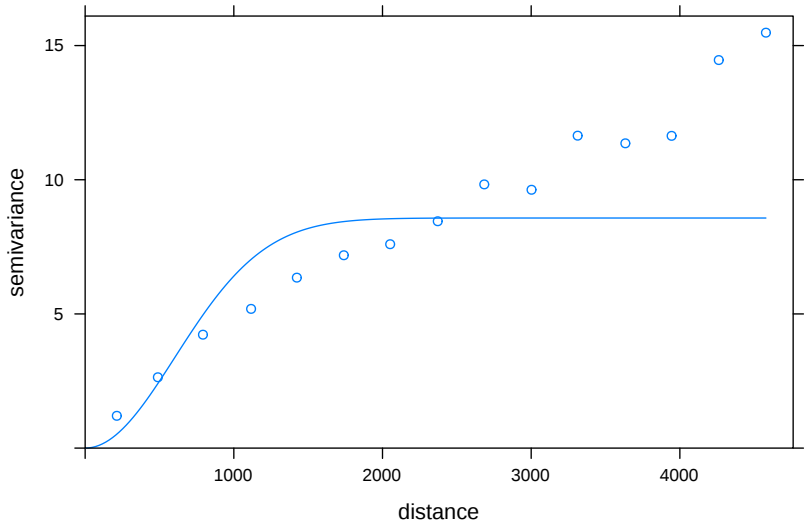

Rycina 7.15.: Model gaussowski zmiennej temp.

Dopasowanie tego modelu może również nastąpić używając funkcji `fit.variogram()` (rycina 7.16).

```
fitted_gau = fit.variogram(vario, model_gau)
fitted_gau
```

```
##   model    psill    range
## 1   Gau 8.573835 852.2404
```

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




Rycina 7.16.: Automatycznie dopasowany model gaussowski zmiennej temp.

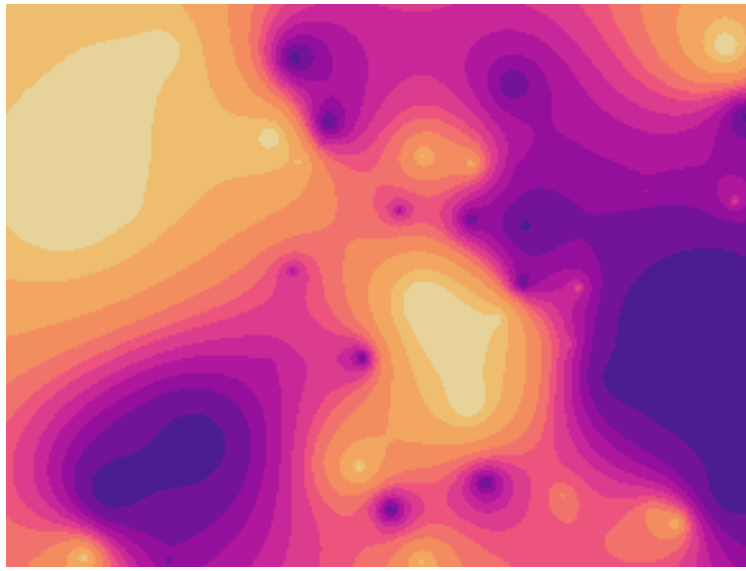
### 7.4.4. Model potęgowy

Model potęgowy (`Pow`) to przykład tzw. modelu nieograniczonego (rycina 7.17). Jego wartość rośnie w nieskończoność, dlatego niemożliwe jest określenie jego zasięgu. W przypadku modelu potęgowego, parametr `range` oznacza wykładnik potęgowy.

Model potęgowy jest określany skrótem `"Pow"` (ryciny 7.18 i 7.19).

```
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
```



Rycina 7.17.: Przykład zjawiska reprezentowanego poprzez model potęgowy.

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

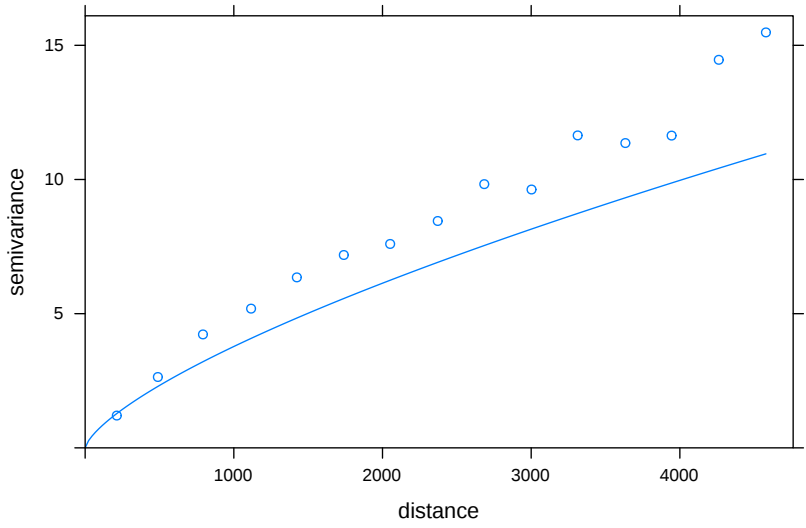
```
fitted_pow = fit.variogram(vario, model_pow)
fitted_pow
```

```
##   model      psill     range
## 1   Pow 0.02515946 0.7535889
```

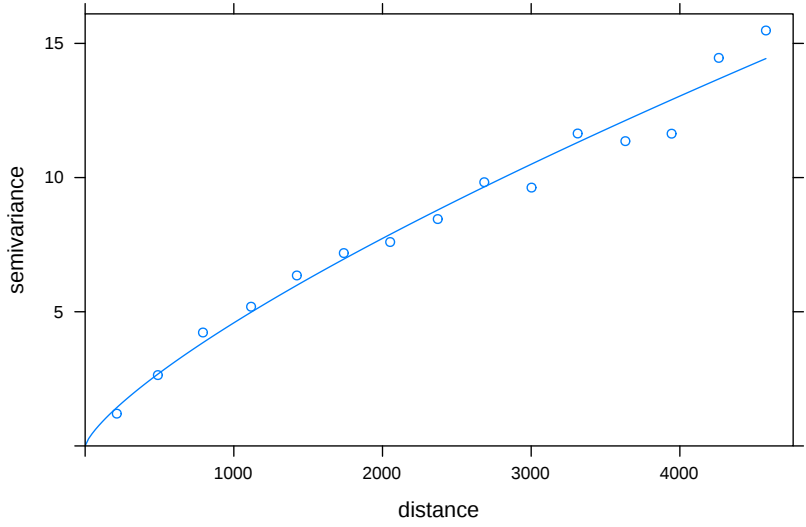
```
plot(vario, model = fitted_pow)
```

### 7.4.5. Porównanie modeli

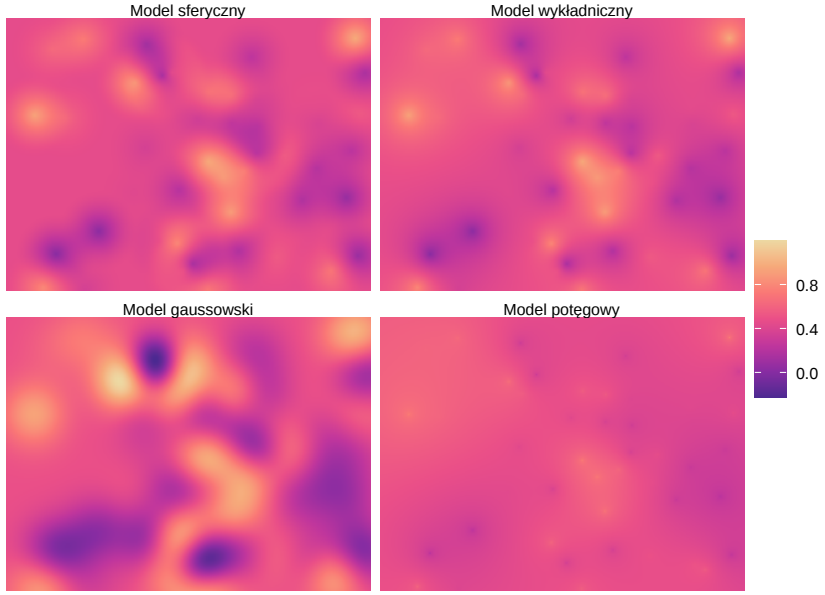
Z uwagi na swoją charakterystykę, każdy z powyższych modeli ma inny zakres wartości. Aby porównać te modele należy je przedstawić używając tej samej skali kolorystycznej (7.20).




Rycina 7.18.: Model potęgowy zmiennej temp.




Rycina 7.19.: Automatycznie dopasowany model potęgowy zmiennej temp.




Rycina 7.20.: Porównanie powierzchni reprezentowanej przez modele sferyczny, wykładniczy, gaussowski i potęgowy.

### 7.4.6. Modele złożone I

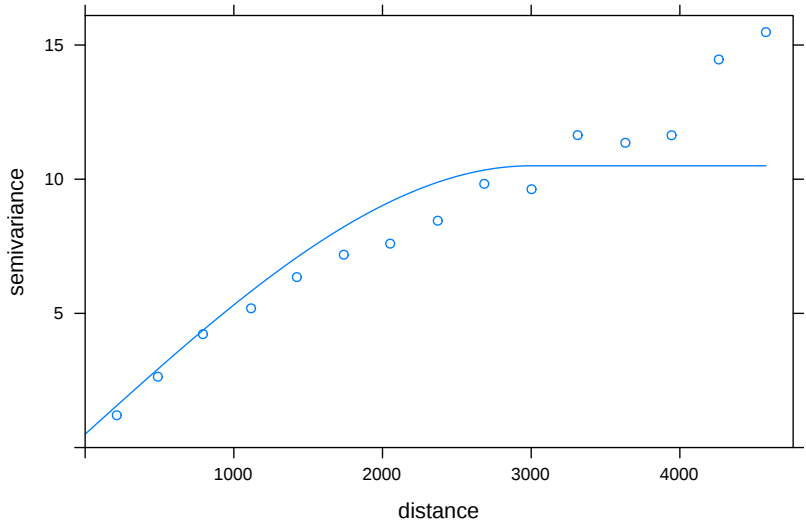
Najczęściej pojedynczy model nie jest w stanie odwzorować dokładnie zmienności przestrzennej analizowanej cechy. W takich sytuacjach konieczne jest połączenie dwóch lub więcej modeli podstawowych. Najbardziej powszechny model złożony składa się z funkcji nuggetowej (dla odległości zero) oraz drugiej funkcji (dla dalszej odległości) (rycina 7.21). 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)
```

Dalsze dopasowanie modeli złożonych również można uzyskać używając funkcji `fit.variogram()` (rycina 7.22).




Rycina 7.21.: Złożony model zmiennej temp.

```
fitted_zl1 = fit.variogram(vario, model_zl1)
fitted_zl1
```

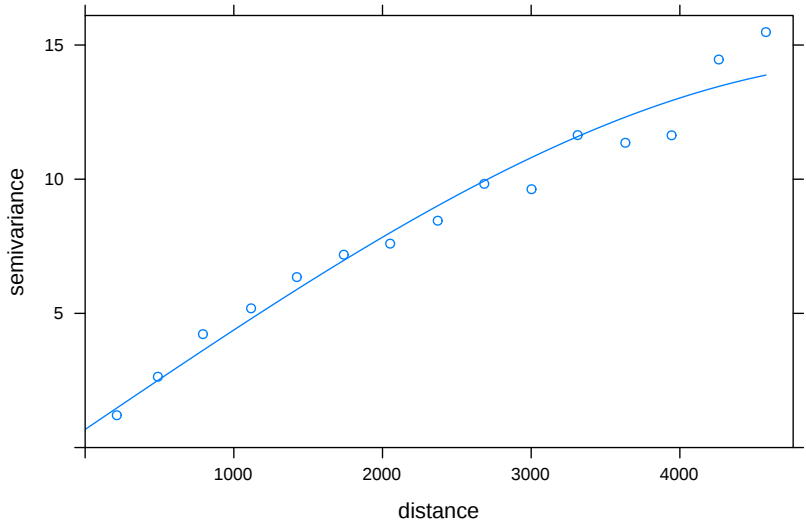
```
##   model      psill    range
## 1   Nug  0.6751142    0.000
## 2   Sph 13.7617233 5511.173
```

```
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 (ryciny 7.23 i 7.24).

```
vario = variogram(temp ~ 1, locations = punkty)
model_zl2 = vgm(10, "Gau", 3000,
                add.to = vgm(4, model = "Gau",
```




Rycina 7.22.: Automatycznie dopasowany złożony model zmiennej temp.

```
                                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
```

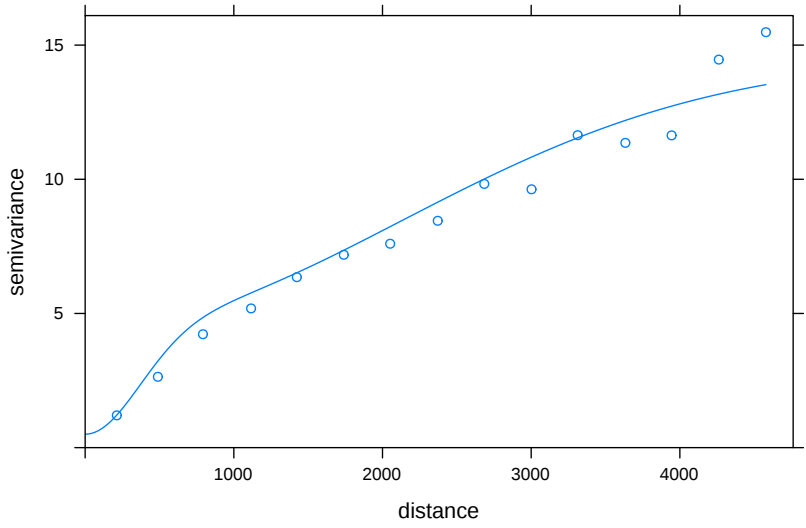
```
plot(vario, model = model_zl2)
```

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

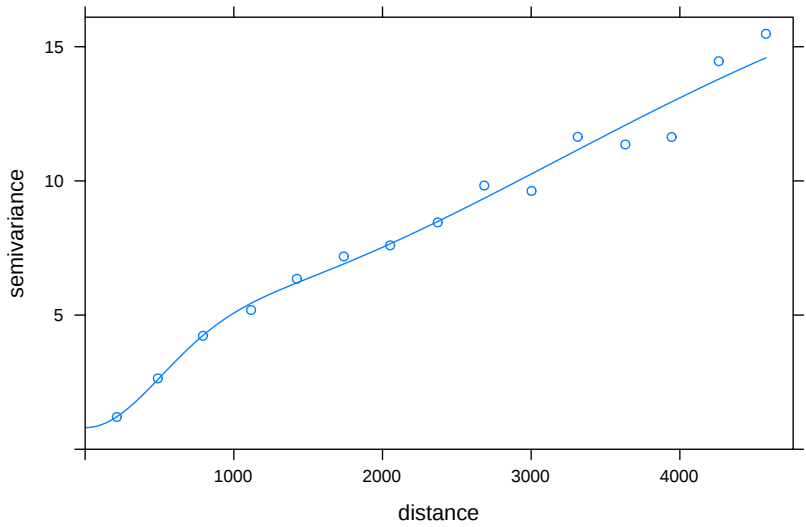
## 7.5. Modelowanie anizotropowe

### 7.5.1. Anizotropia

Uwzględnienie anizotropii wymaga zamiany parametru zasięgu na trzy inne parametry (rycina 7.25):

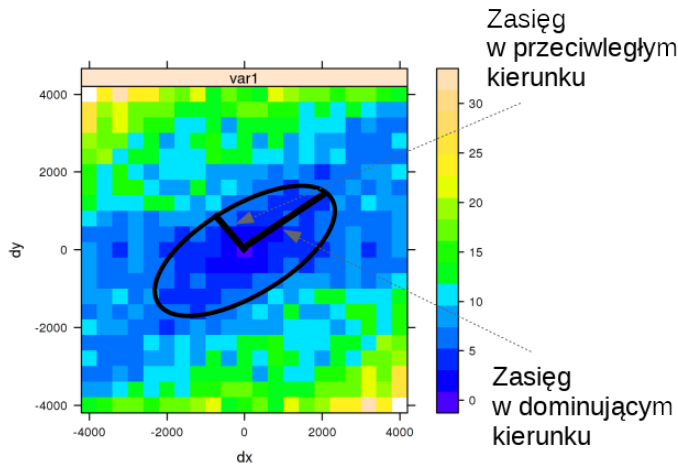



Rycina 7.23.: Złożony model zmiennej temp składający się z modelu nuggetowego oraz dwóch modeli gaussowskich.




Rycina 7.24.: Automatycznie dopasowany złożony model zmiennej temp składający się z modelu nuggetowego oraz dwóch modeli gaussowskich.

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

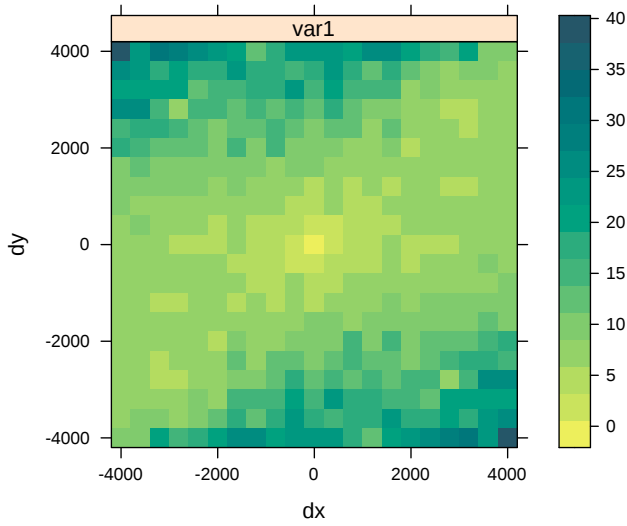



Rycina 7.25.: Podstawowe parametry mapy semiwariogramu.

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) (rycina 7.26).

```
vario_map = variogram(temp ~ 1,
                      locations = punkty,
                      cutoff = 4000,
                      width = 400,
                      map = TRUE)
plot(vario_map, threshold = 30,
     col.regions = hcl.colors(40, palette = "ag_GrnYl", rev = TRUE))
```

Anizotropia może być także reprezentowana używając semiwariogramów kierunkowych (rycina 7.27)



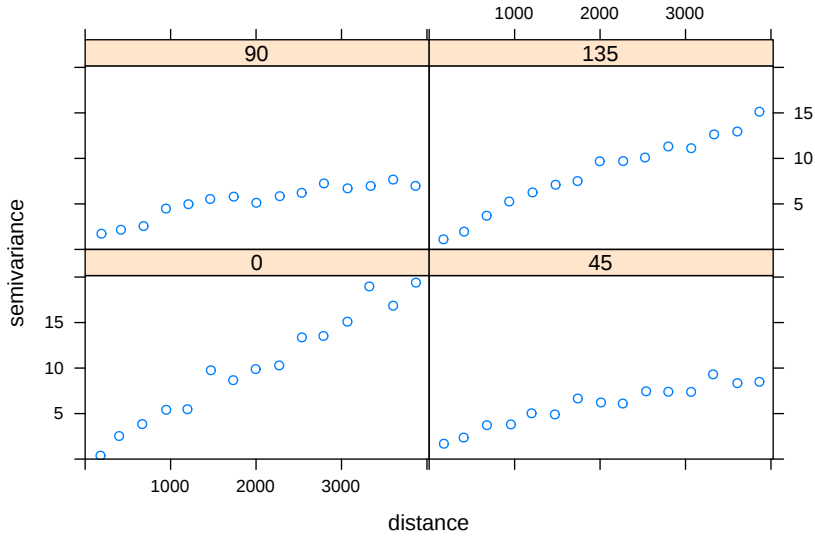

Rycina 7.26.: Mapa semiwariogramu zmiennej temp.

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

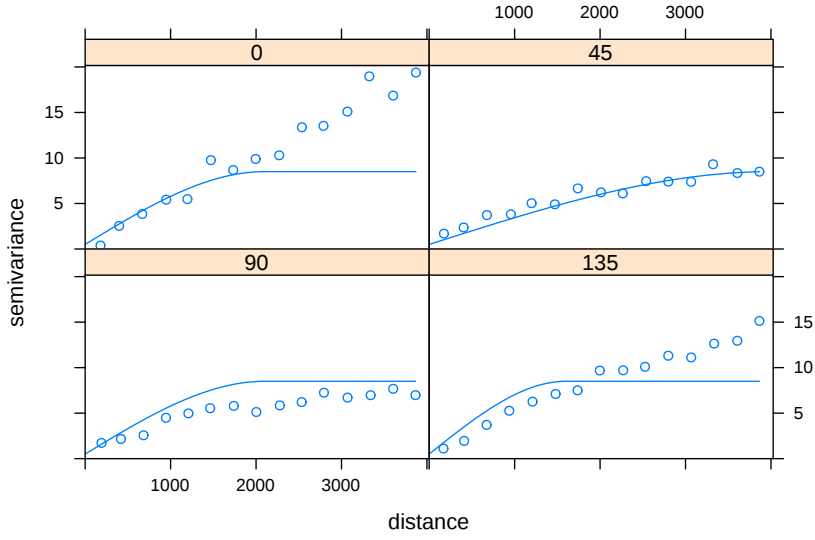
Następnie semiwariogramy kierunkowe mogą być modelowane ręcznie (rycina 7.28) lub też używając automatycznego dopasowania (rycina 7.29).

```
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)
```

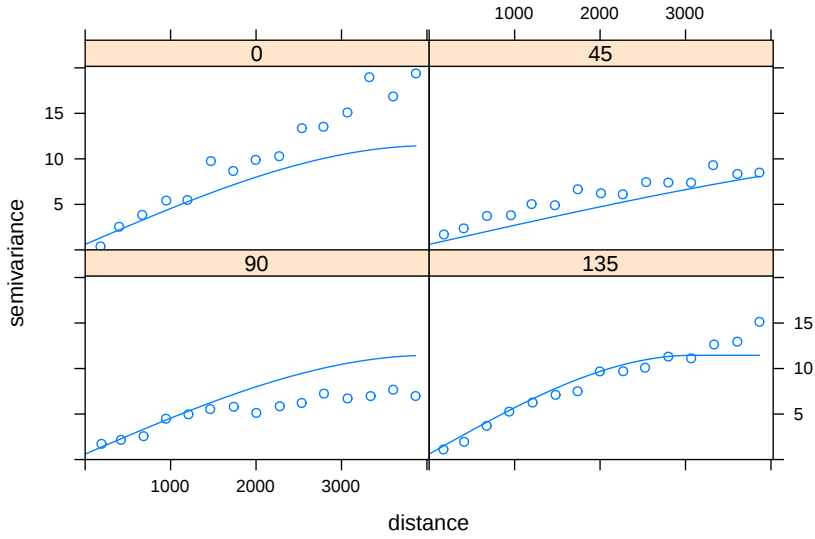
```
vario_kier_fit2 = fit.variogram(vario_kier,
                                vgm(model = "Sph",
                                    anis = c(45, 0.4),
                                    nugget = 0.5))
plot(vario_kier, vario_kier_fit2, as.table = TRUE)
```




Rycina 7.27.: Semiwariogramy kierunkowe zmiennej temp.




Rycina 7.28.: Modele kierunkowe zmiennej temp.




Rycina 7.29.: Automatycznie dopasowane modele kierunkowe zmiennej temp.

## 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(sf)
library(stars)
library(gstat)
library(geostatbook)
data(punkty)
data(siatka)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 8.1. Kriging

### 8.1.1. Interpolacja geostatystyczna

Kriging (interpolacja geostatystyczna) to grupa metod estymacji zaproponowana w latach 50. przez Daniego 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 metod krigingu, w tym:

- Kriging prosty (ang. *Simple kriging*) (sekcja 8.2)
- Kriging zwykły (ang. *Ordinary kriging*) (sekcja 8.3)
- Kriging z trendem (ang. *Kriging with a trend*) (sekcja 8.4)
- Kriging stratyfikowany (ang. *Kriging within strata* – KWS)
- Kriging prosty ze zmiennymi średnimi lokalnymi (ang. *Simple kriging with varying local means* - SKlm) (sekcja 9.1)
- Kriging z zewnętrznym trendem/Uniwersalny kriging (ang.*Kriging with an external trend/Universal kriging*) (sekcja 9.2)
- Kokriging (ang. *Co-kriging*) (sekcja 10.1)
- Kriging danych kodowanych (ang. *Indicator kriging*) (sekcja 11.1)

- Inne

## 8.2. Kriging prosty

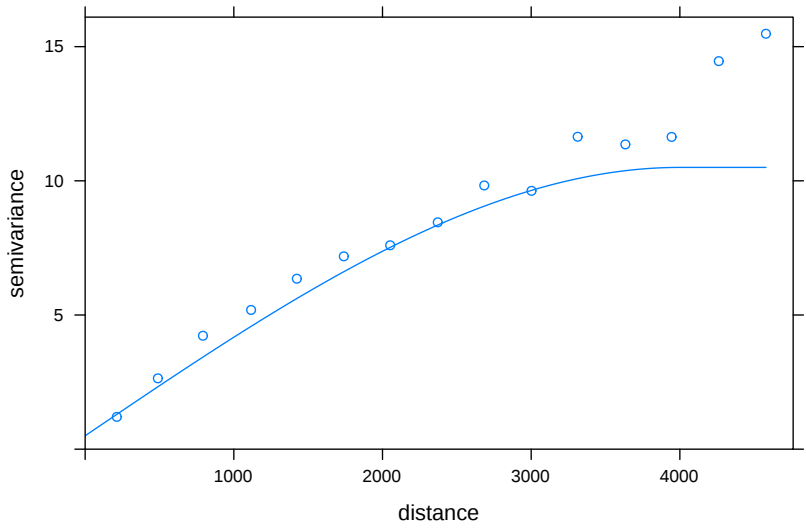
### 8.2.1. Kriging prosty (ang. *Simple kriging*)

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 (rycina 8.1).

```
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)
```




Rycina 8.1.: Model złożony z modelu nuggetowego i sferycznego dla zmiennej temp.

```
# 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] 15.22251
```

```
sk = krige(temp ~ 1,
            locations = punkty,
            newdata = siatka,
            model = model,
            beta = 15)
```

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

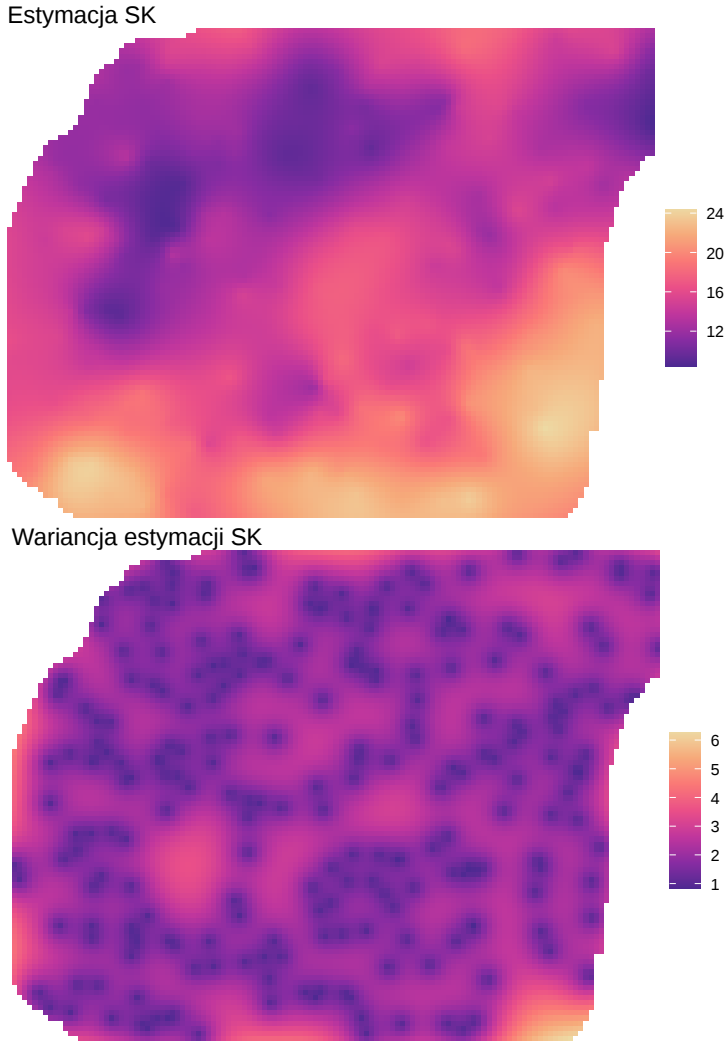
Wynik krigingu prostego, jak i każdy inny uzyskany z użyciem pakietu **gstat**, można podejrzeć wpisując nazwę wynikowego obiektu. Szczególnie ważne są dwie, nowe zmienne - `var1.pred` oraz `var1.var`. Pierwsza z nich oznacza wartość estymowaną dla każdego oczka siatki, druga zaś mówi o wariancji estymacji.

```
sk
```

```
## stars object with 2 dimensions and 2 attributes
## attribute(s):
##                Min.  1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## var1.pred  8.473052 12.70587 14.961083 15.488215 17.711775 24.36355 1270
## var1.var   0.821095  1.61326  1.974793  2.041073  2.355011  6.26745 1270
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92    NA   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92    NA   NULL [y]
```

Obie uzyskane zmienne można wyświetlić z użyciem funkcji `plot()` (rycina 8.2).

```
plot(sk["var1.pred"], col = paleta)
plot(sk["var1.var"], col = paleta)
```




Rycina 8.2.: Estymacja i wariancja estymacji używając metody krigingu prostego (SK).

# 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)
```

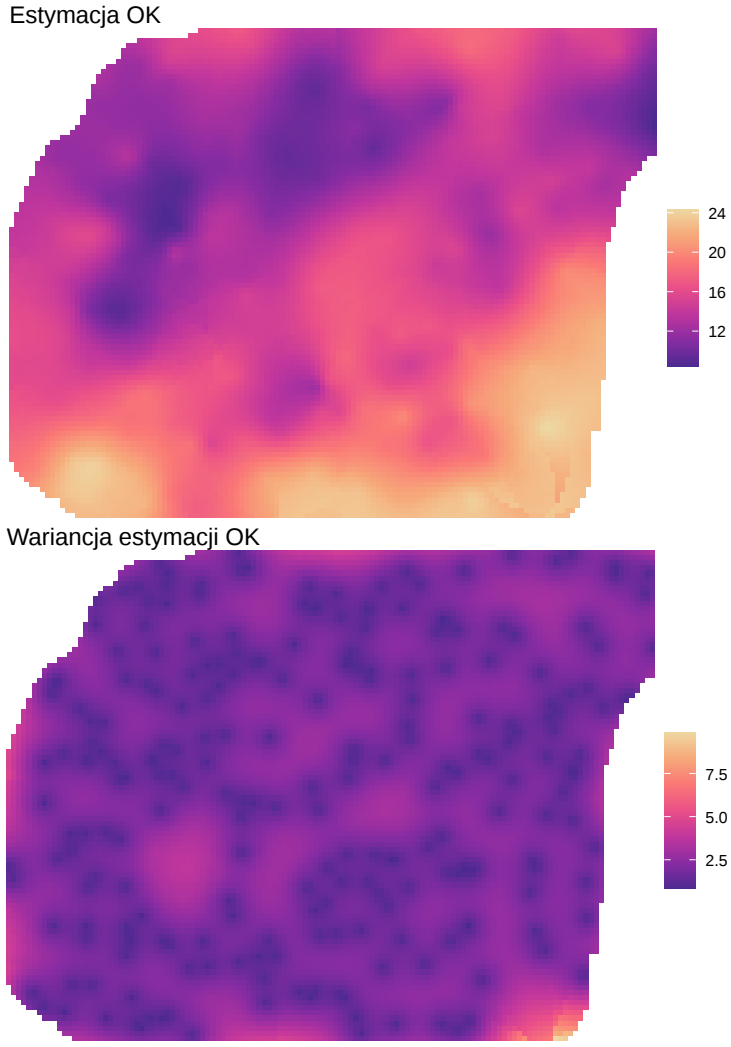
```
## [using ordinary kriging]
```

Podobnie jak w przypadku krigingu prostego, można przyjrzeć się wynikom estymacji podając nazwę wynikowego obiektu oraz wyświetlić je używając funkcji `plot()` (rycina 8.3).

```
ok
```

```
## stars object with 2 dimensions and 2 attributes
## attribute(s):
##                 Min.  1st Qu.    Median      Mean  3rd Qu.      Max. NA's
## var1.pred  8.4763353 12.71717 14.967146 15.531098 17.70921 24.326622 1270
## var1.var   0.8215425  1.62057  1.990971  2.082008  2.39254  9.882173 1270
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92    NA   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92    NA   NULL [y]
```

```
plot(ok["var1.pred"], col = paleta)
plot(ok["var1.var"], col = paleta)
```




Rycina 8.3.: Estymacja i wariancja estymacji używając metody krigingu zwykłego (OK).

# 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. W pierwszym kroku konieczne jest dodanie współrzędnych do używanego obiektu i do używanej siatki.

```
# dodanie współrzędnych do punktów
punkty$x = st_coordinates(punkty)[, 1]
punkty$y = st_coordinates(punkty)[, 2]
# dodanie współrzędnych do siatki
siatka$x = st_coordinates(siatka)[, 1]
siatka$y = st_coordinates(siatka)[, 2]
```

Współrzędne dodawane są do całej (regularnej) siatki. Możemy przyciąć je do badanego obszaru poprzez wpisanie wartości `NA` w miejscach poza naszym obszarem zainteresowań.

```
siatka$x[is.na(siatka$X2)] = NA
siatka$y[is.na(siatka$X2)] = NA
```

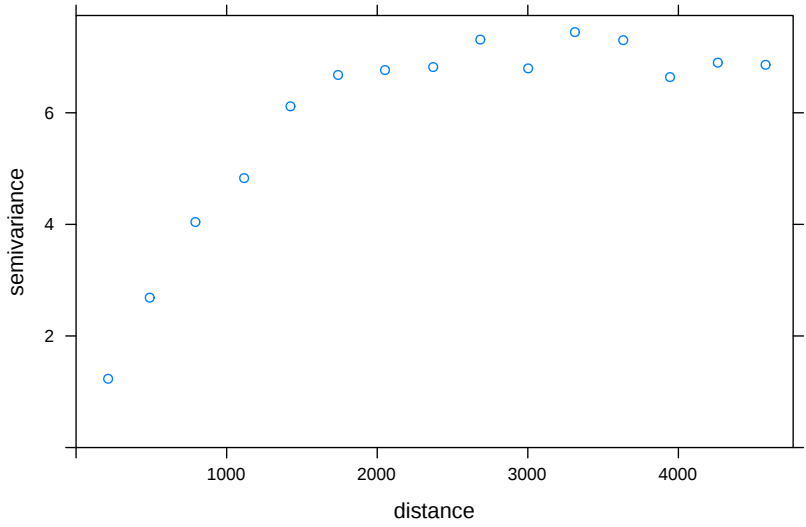
Następnie pierwszy z argumentów w funkcji `variogram()` musi przyjąć postać `temp ~ x + y`, co oznacza, że uwzględniamy liniowy trend zależny od współrzędnej `x` oraz `y` (rycina 8.4).

```
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 (rycina 8.5).

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

```
##   model     psill    range
## 1   Nug 0.2568762    0.000
## 2   Sph 6.7107464 2087.465
```




Rycina 8.4.: Semiwariogram zmiennej temp uwzględniający współrzędne x i y.

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

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

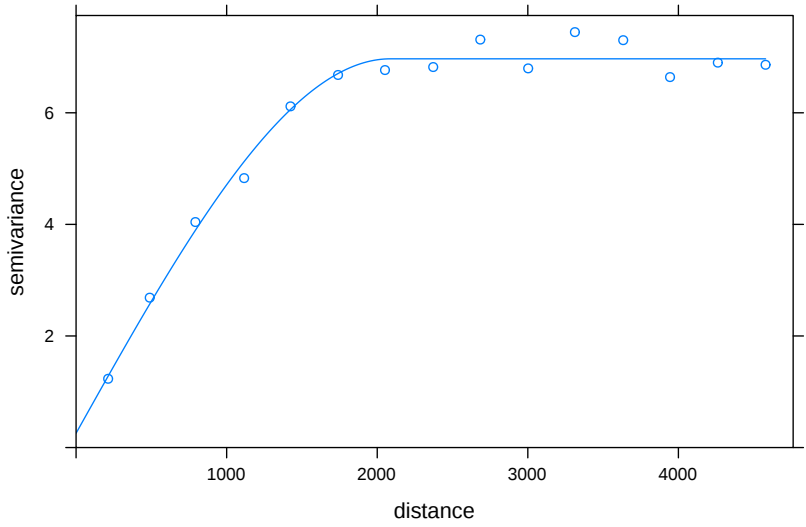
```
## [using universal kriging]
```

Wyświelenie wyników odbywa się używając funkcji `plot()`.

```
plot(kzt["var1.pred"], col = paleta)
plot(kzt["var1.var"], col = paleta)
```

## 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 (rycina 8.7). W rozdziałach 10 oraz 9 pokazane będą uzyskane wyniki interpolacji temperatury powietrza korzystając z innych metod krigingu.




Rycina 8.5.: Model semiwariogramu zmiennej temp uwzględniający współrzędne x i y.

## 8.6. Zadania

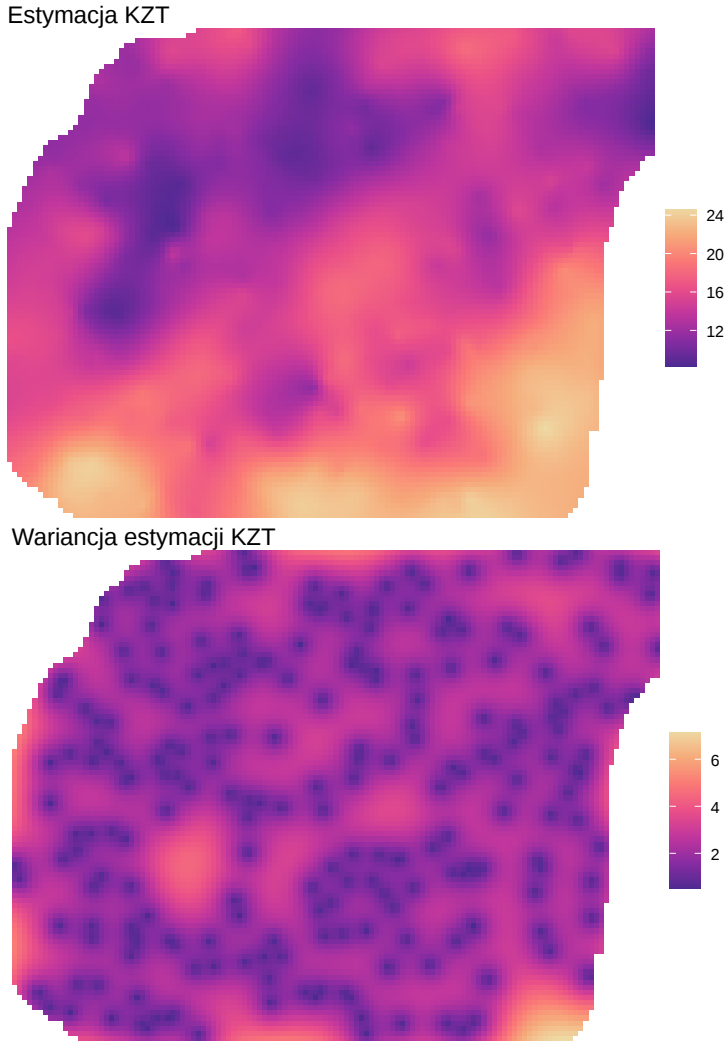
Zadania w tym rozdziale są oparte o dane z obiektu `punkty_pref`. Możesz go wczytać używając poniższego kodu:

```
data(punkty_pref)
```

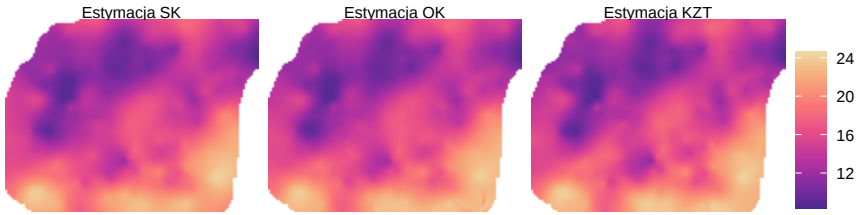
Na jego podstawie stwórz trzy obiekty - `punkty_pref1` zawierający wszystkie punkty, `punkty_pref2` zawierający losowe 100 punktów, oraz `punkty_pref3` zawierający losowe 30 punktów.

```
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), ]
```

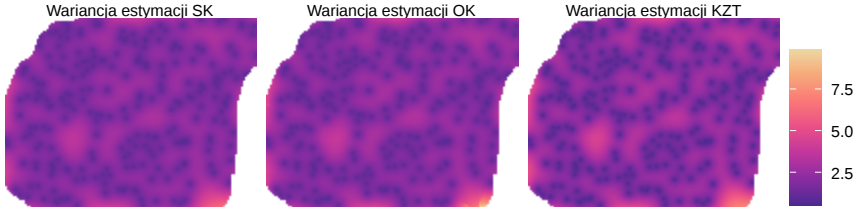
1. Zbuduj optymalne modele semiwariogramu zmiennej `srtm` dla trzech zbiorów danych - `punkty_pref1`, `punkty_pref2`, `punkty_pref3`. Porównaj graficznie uzyskane modele.
2. W oparciu o uzyskane modele stwórz estymacje zmiennej `srtm` dla trzech zbiorów danych - `punkty_pref1`, `punkty_pref2`, `punkty_pref3` używając krigingu prostego. Porównaj graficznie zarówno mapy estymacji jak i mapy wariancji. Opisz zaobserwowane różnice.




Rycina 8.6.: Estymacja i wariancja estymacji używając metody krigingu z trendem (KZT).




Rycina 8.7.: Porównanie wyników estymacji używając metody krigingu prostego (SK), krigingu zwykłego (OK) i krigingu z trendem (KZT).




Rycina 8.8.: Porównanie wariancji estymacji używając metody krigingu prostego (SK), krigingu zwykłego (OK) i krigingu z trendem (KZT).

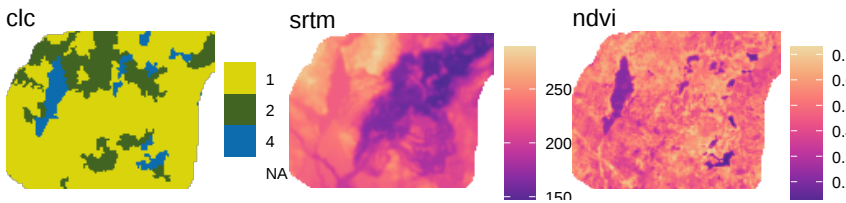
3. W oparciu o uzyskane modele stwórz estymacje zmiennej `srtm` dla zbioru danych `punkty_pref3` używając krigingu zwykłego. Sprawdź jak wygląda wynik estymacji uwzględniając (i) 10 najbliższych obserwacji, (ii) 30 najbliższych obserwacji, (iii) obserwacje w odległości do 2 km.
4. Używając krigingu z 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?
6. Dla zmiennej `temp` z obiektu `punkty_pref1` stwórz mapę semiwariogramu. Czy ta zmienna wykazuje anizotropię przestrzenną? Jeżeli tak to stwórz semiwariogramy kierunkowe i ich modele, a następnie estymację zmiennej `temp`.

# 9. Estymacje używające danych uzupełniających

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

```
library(sf)
library(stars)
library(gstat)
library(geostatbook)
data(punkty)
data(siatka)
data(dane_uzup)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

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, kriging prosty ze zmiennymi średnimi lokalnymi, czy kriging uniwersalny.



## 9.1. Kriging prosty ze zmiennymi średnimi lokalnymi (LVM)

### 9.1.1. Kriging prosty ze zmiennymi średnimi lokalnymi (LVM) (ang. *Simple kriging with varying local means*)

Kriging prosty 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
## 17.41296753 -0.01021971
```

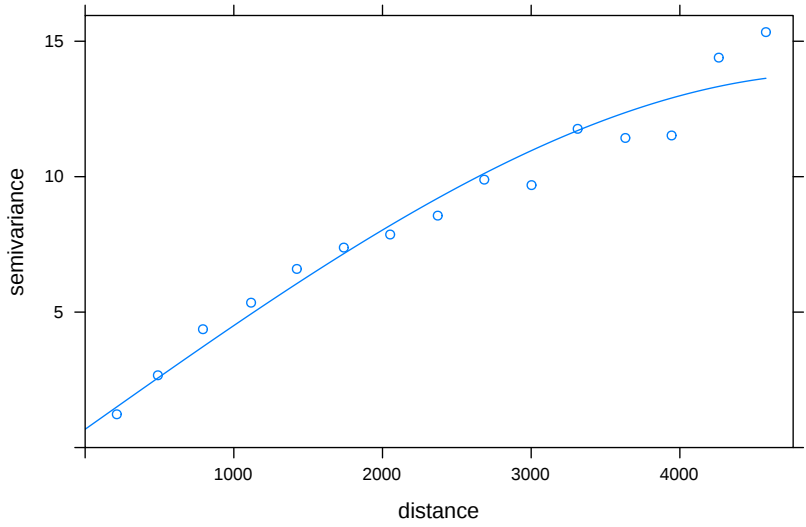
Wykorzystując relację pomiędzy tymi dwoma zmiennymi tworzony jest semiwariogram empiryczny, który następnie jest modelowany (rycina 9.1).

```
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  0.6756367    0.000
## 2   Sph 13.1450299 5085.204
```

```
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.




Rycina 9.1.: Model semiwariogramu zmiennej temp używając zmiennej srtm.

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

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

```
sk_lvm
```

```
## stars object with 2 dimensions and 2 attributes
## attribute(s):
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## var1.pred  8.559515 12.745494 14.958937 15.523648 17.759209 24.240945 1242
## var1.var   1.066204  1.875899  2.248138  2.321278  2.638222  6.852509 1242
## dimension(s):
##   from  to offset delta                refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92    NA   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92    NA   NULL [y]
```

```
plot(sk_lvm["var1.pred"], col = paleta)
plot(sk_lvm["var1.var"], col = paleta)
```

## 9.2. Kriging uniwersalny

### 9.2.1. 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 (ryciny 9.3, 9.4, 9.5).

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

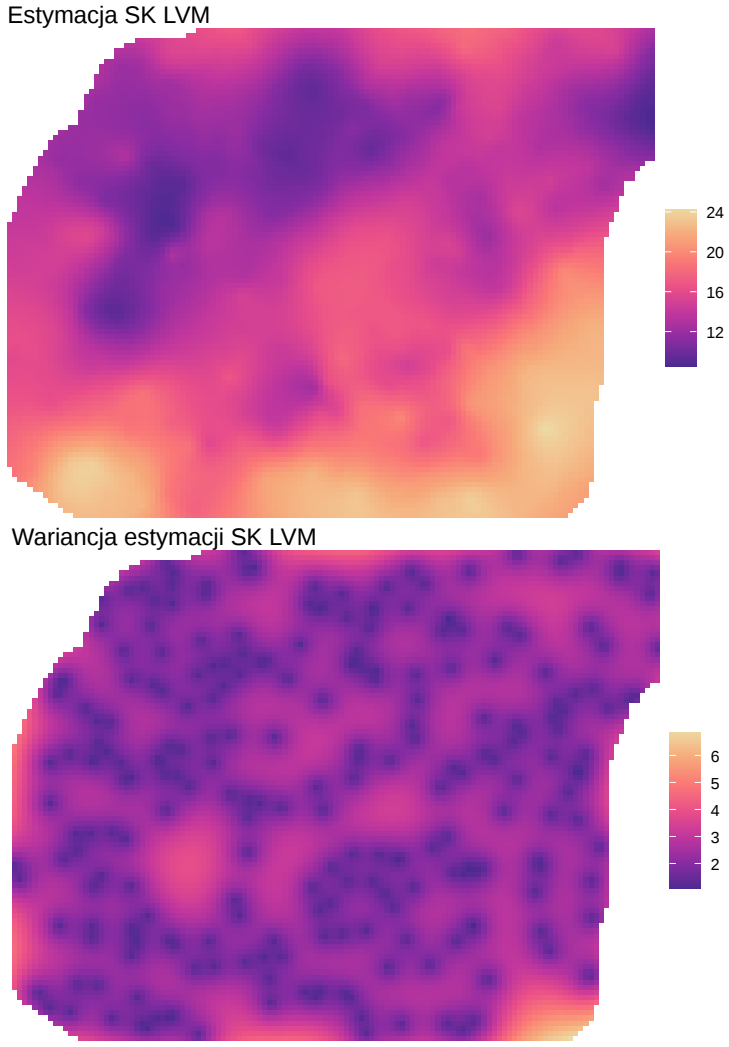
```
##   model    psill    range
## 1   Nug 1.626245    0.000
## 2   Sph 9.059005 6426.143
```

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

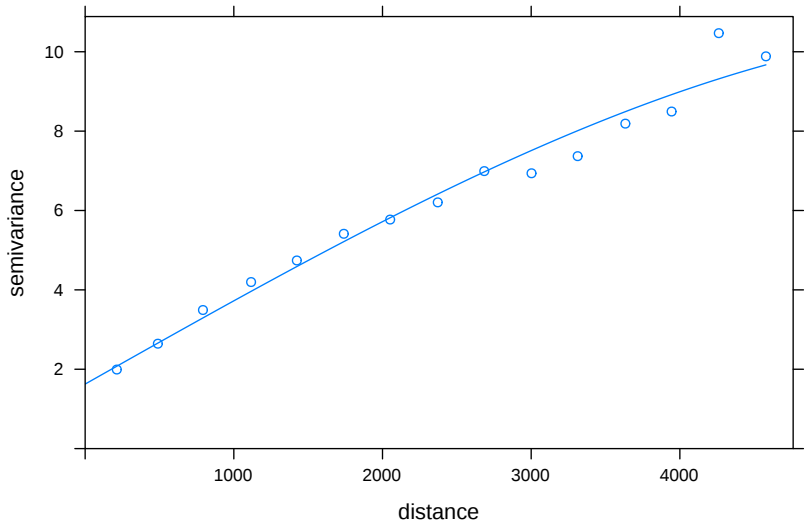
```
dane_uzup$clc = as.factor(dane_uzup$clc)
plot(dane_uzup["clc"], col = c("#d9d40c", "#416422", "#0c6cae"))
```

```
uk1 = krige(temp ~ clc,
            locations = punkty,
            newdata = dane_uzup,
            model = vario_fit_uk1)
```

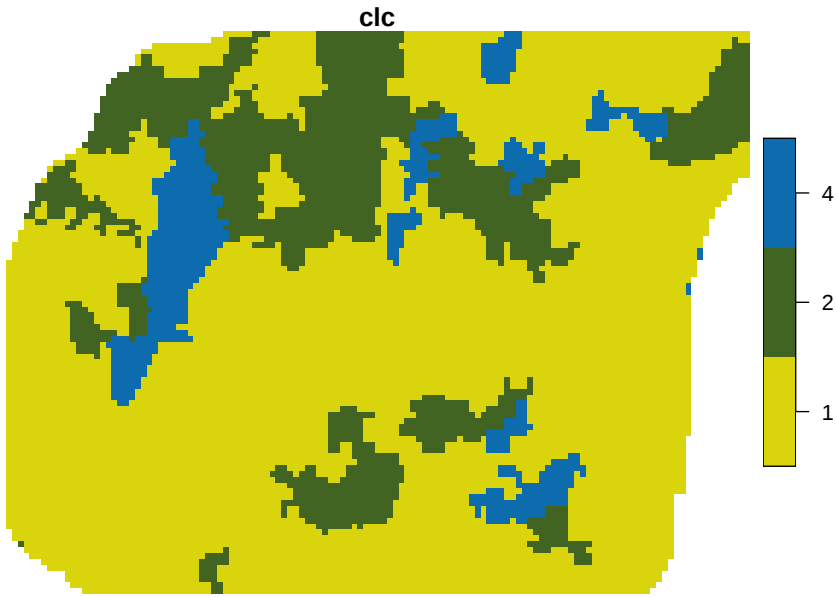
```
## [using universal kriging]
```




Rycina 9.2.: Estymacja i wariancja estymacji używając metody prostego krigingu ze zmiennymi średnimi lokalnymi (LVM).




Rycina 9.3.: Model semiwariogramu zmiennej temp używając zmiennej clc.




Rycina 9.4.: Rozkład przestrzenny wartości zmiennej clc używanej w modelu.

```
plot(uk1["var1.pred"], col = paleta)
plot(uk1["var1.var"], col = paleta)
```

W kolejnym przykładzie zastosowane są już dwie zmienne uzupełniające - wartość wskaźnika wegetacji (`ndvi`) oraz wysokość nad poziomem morza (`srtm`) (ryciny 9.6, 9.7).

```
vario_uk2 = variogram(temp ~ ndvi + srtm, location = punkty)
# vario_uk2
# 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  0.7602125    0.000
## 2   Sph 12.4326154 5188.676
```

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

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

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

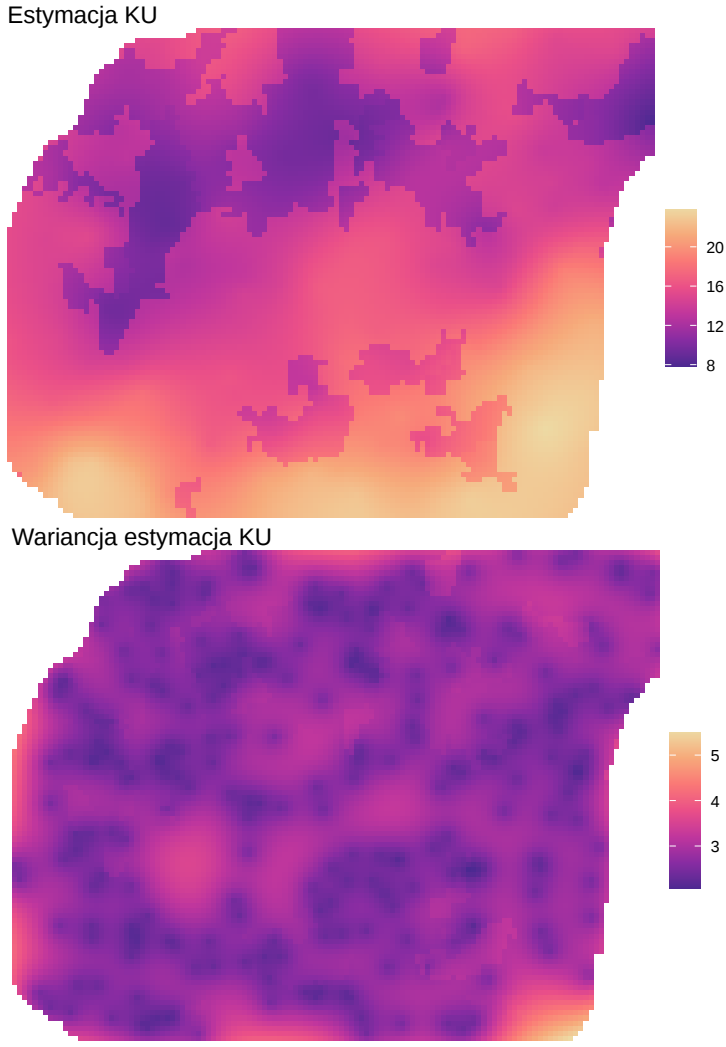
```
plot(uk2["var1.pred"], col = paleta)
plot(uk2["var1.var"], col = paleta)
```

## 9.3. Zadania

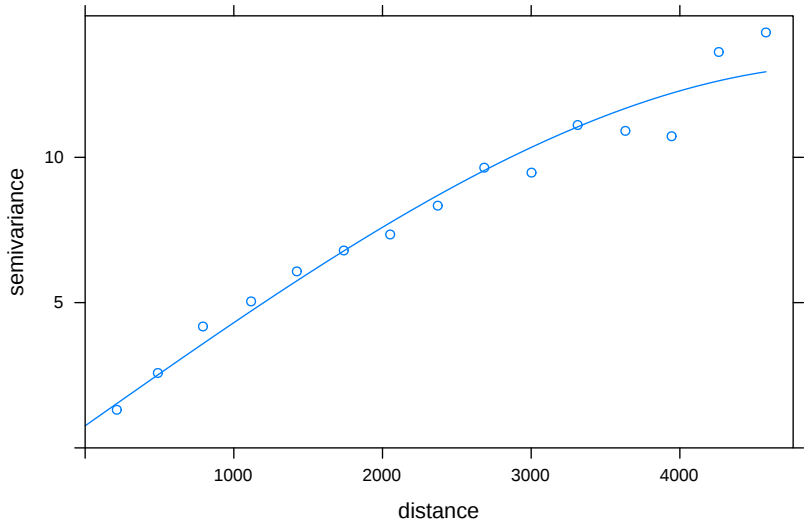
Zadania w tym rozdziale są oparte o dane z obiektu `punkty`.

```
data(punkty)
```

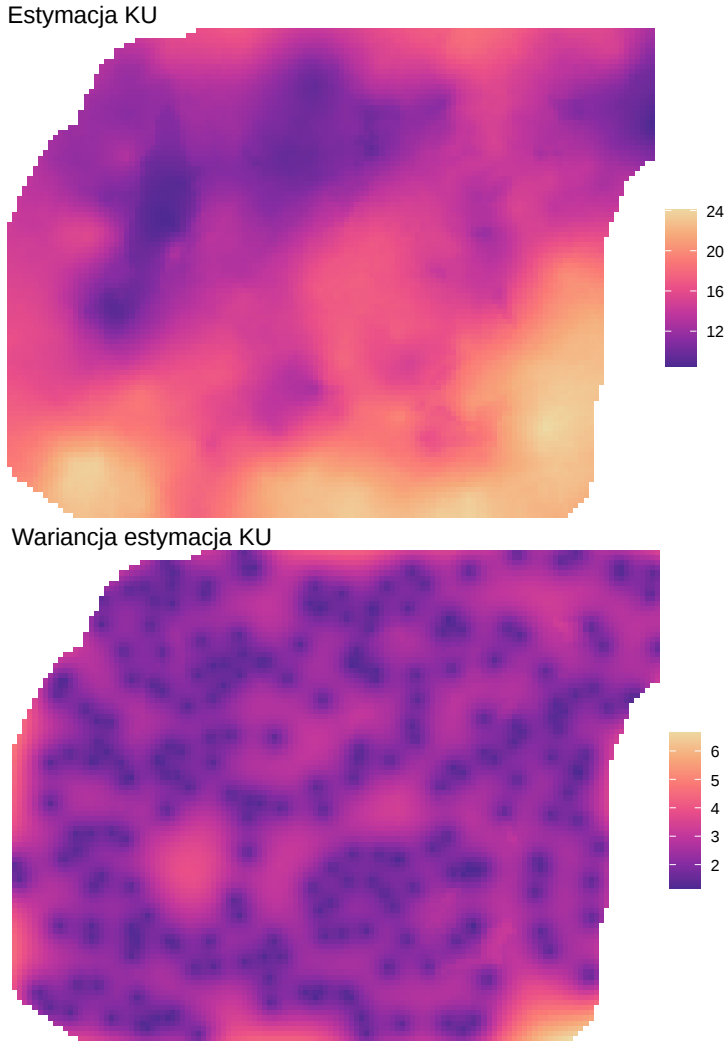
1. Zastosuj kriging prosty ze zmiennymi średnimi lokalnymi do stworzenia estymacji zmiennej `ndvi` używając jej relacji ze zmienną `savi`.
2. Stwórz estymację krigingu uniwersalnego dla zmiennej `ndvi` używając jej relacji ze zmienną `savi`.
3. Stwórz estymację krigingu uniwersalnego dla zmiennej `ndvi` używając jej relacji ze zmiennymi `clc`, `srtm`, `temp` i `savi`.
4. Porównaj graficznie trzy powyższe estymacje. Opisz podobieństwa i różnice.




Rycina 9.5.: Estymacja i wariancja estymacji używając zmiennej clc i metody krigingu uniwersalnego (KU).




Rycina 9.6.: Model semiwariogramu zmiennej temp używając zmiennych ndvi i srtm.




Rycina 9.7.: Estymacja i wariancja estymacji używając zmiennych ndvi i srtm i metody krigingu uniwersalnego (KU).

# 10. Estymacje wielozmienne

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

```
library(sf)
library(stars)
library(gstat)
library(geostatbook)
data(punkty)
data(punkty_ndvi)
data(siatka)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 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

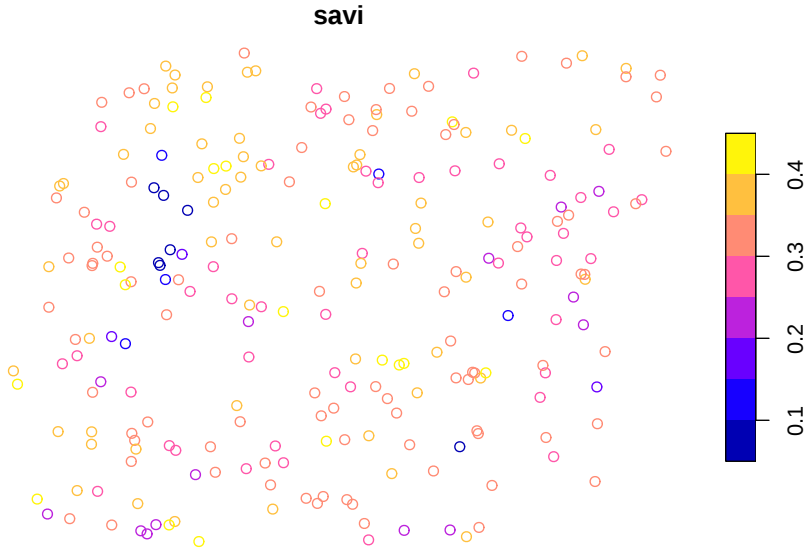
## 10.2. Krossemiwariogramy

### 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.

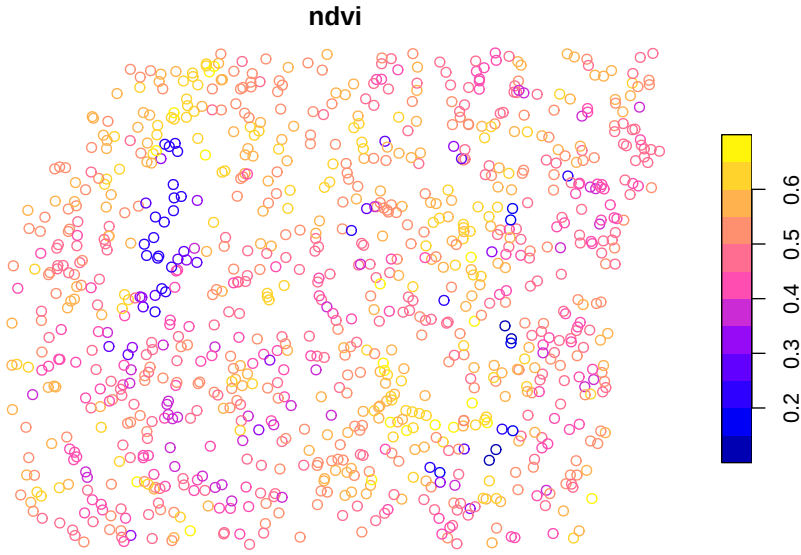
W poniższym przykładzie istnieją dwie zmienne, `savi` ze zbioru `punkty` pomierzona w 242 lokalizacjach oraz `ndvi` ze zbioru `punkty_ndvi` pomierzona w 992 punktach (ryciny 10.1 i 10.1).

```
plot(punkty["savi"])
```




Rycina 10.1.: Mapa wartości zmiennej savi.

```
plot(punkty_ndvi["ndvi"])
```




Rycina 10.2.: Mapa wartości zmiennej 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.

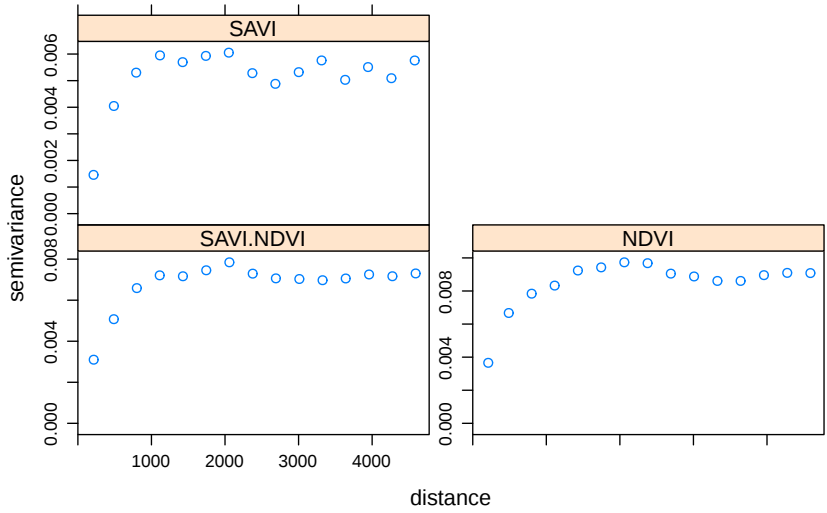
```
g = gstat(NULL,
          id = "SAVI",
          form = savi ~ 1,
          data = punkty)
g = gstat(g,
          id = "NDVI",
          form = ndvi ~ 1,
          data = punkty_ndvi)
g
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 242 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 992 x 1
```

Z uzyskanego w ten sposób obiektu tworzymy krossemiwariogram (funkcja

`variogram()`), a następnie go wizualizujemy używając funkcji `plot()` (rycina 10.3).

```
v = variogram(g)
plot(v)
```




Rycina 10.3.: Krossemiwariogram zmiennych savi i ndvi.

## 10.3. Modelowanie krossemiwariogramów

Modelowanie krossemiwariogramów, podobnie jak ich tworzenie, odbywa się używając funkcji `gstat()`, gdzie podaje się 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 = 1200, nugget = 0.001)
g1 = gstat(g, model = g_model, fill.all = TRUE)
g1
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 242 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 992 x 1
```

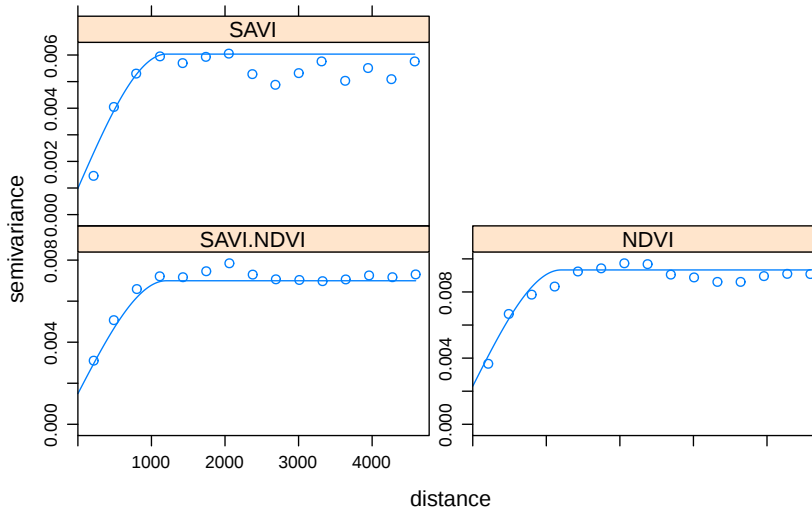
```
## variograms:
##               model psill range
## SAVI[1]         Nug 0.001     0
## SAVI[2]         Sph 0.006  1200
## NDVI[1]         Nug 0.001     0
## NDVI[2]         Sph 0.006  1200
## SAVI.NDVI[1]    Nug 0.001     0
## SAVI.NDVI[2]    Sph 0.006  1200
```

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 (rycina 10.4). Funkcja `fit.lmc()` oczekuje co najmniej dwóch elementów, krossemiwariogramu oraz modelów krossemiwariancji. W poniższym przykładzie dodatkowo użyto parametru `correct.diagonal = 1.01`, z uwagi na to że analizowane zmienne wykazywały bardzo silną korelację oraz parametru `fit.method`. Ten ostatni parametr określa, która z użytych metod automatycznego dopasowania semiwariogramów jest używana. Pakiet **gstat** ma domyślnie ustawione `fit.method = 7`, co daje największe znaczenie parom punktów o najmniejszym zasięgu. Ta opcja nie jest zazwyczaj optymalna w przypadku krossemiwariogramów - tutaj częściej stosuje się `fit.method = 6` (wagi nie są nadawane) lub `fit.method = 1` (wagi proporcjonalne do liczby par punktów w każdym przedziale).

```
g2 = fit.lmc(v, g1,
             correct.diagonal = 1.01,
             fit.method = 6)
g2
```

```
## data:
## SAVI : formula = savi`~`1 ; data dim = 242 x 5
## NDVI : formula = ndvi`~`1 ; data dim = 992 x 1
## variograms:
##               model        psill range
## SAVI[1]         Nug 0.0009873135     0
## SAVI[2]         Sph 0.0050465345  1200
## NDVI[1]         Nug 0.0023006100     0
## NDVI[2]         Sph 0.0070299532  1200
## SAVI.NDVI[1]    Nug 0.0014922022     0
## SAVI.NDVI[2]    Sph 0.0055018027  1200
```

```
plot(v, g2)
```




Rycina 10.4.: Automatycznie dopasowane modele krossemiwariogramu zmiennych savi i ndvi.

## 10.4. Kokriging

Posiadając dopasowane modele oraz siatkę można uzyskać wynik używając funkcji `predict()` (rycina 10.5).

```
ck = predict(g2, 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` - wariancja zmiennej `ndvi`
5. `cov.SAVI.NDVI` - kowariancja zmiennych `savi` oraz `ndvi`

```
ck
```

```
## stars object with 2 dimensions and 5 attributes
## attribute(s):
##                       Min.     1st Qu.      Median        Mean     3rd Qu.
## SAVI.pred      0.077243684 0.290908153 0.321211504 0.315700226 0.348022981
## SAVI.var       0.001525063 0.002379011 0.002638353 0.002652832 0.002905902
## NDVI.pred      0.205549518 0.468630003 0.508090071 0.501150702 0.542080400
## NDVI.var       0.003082640 0.003893078 0.004212984 0.004272496 0.004578181
## cov.SAVI.NDVI  0.001930305 0.002625231 0.002885524 0.002929714 0.003180885
##                        Max. NA's
## SAVI.pred      0.437090600 1270
## SAVI.var       0.004777928 1270
## NDVI.pred      0.658528553 1270
## NDVI.var       0.007378830 1270
## cov.SAVI.NDVI  0.005432433 1270
## dimension(s):
##   from  to offset delta                       refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92 (wit...    NA   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92 (wit...    NA   NULL [y]
```

```
plot(ck["SAVI.pred"], col = paleta)
plot(ck["SAVI.var"], col = paleta)
```

## 10.5. Zadania

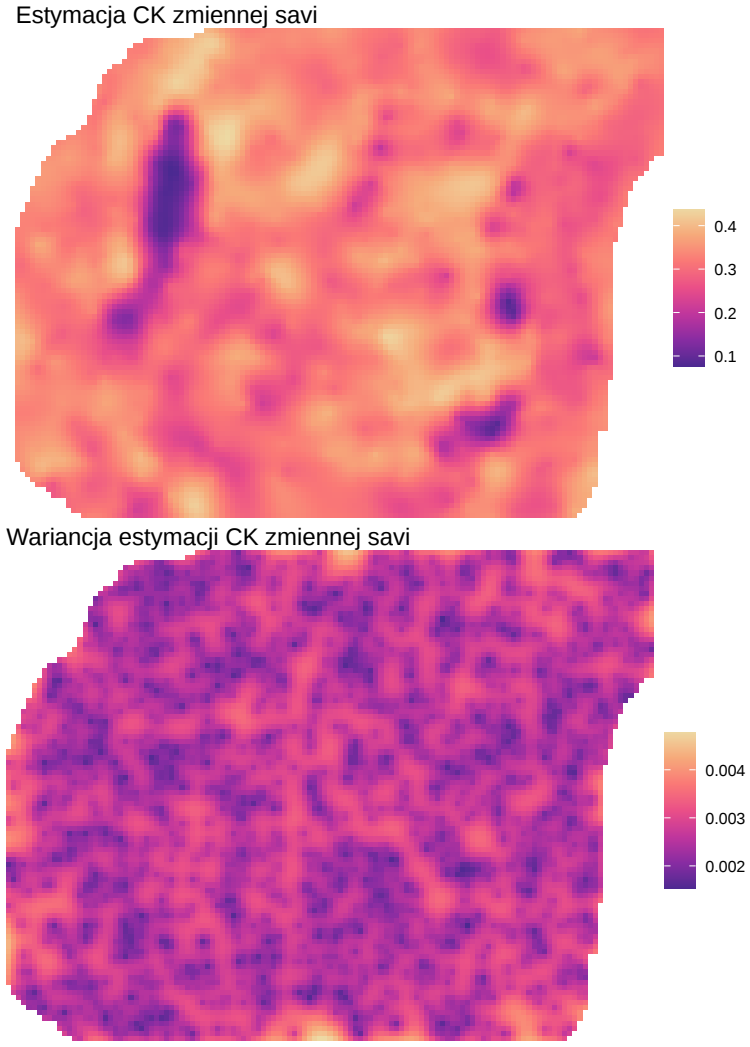
Zadania w tym rozdziale są oparte o dane z `meuse.all` z pakietu **gstat**.

```
data("meuse.all")
meuse = st_as_sf(meuse.all, coords = c("x", "y"))
```

Na jego podstawie wydziel dwa obiekty - `meuse164` zawierający tylko zmienną `cadmium` (kadm) dla 164 punktów, oraz `meuse60` zawierający tylko zmienną `copper` (miedź) dla 60 punktów.

```
set.seed(431)
meuse164 = meuse["cadmium"]
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 `cadmium` dla obiektu `meuse164` oraz zmiennej `copper` dla obiektu `meuse60`. Porównaj graficznie uzyskane modele.




Rycina 10.5.: Estymacja i wariancja estymacji zmiennej savi używając metody kokrigingu (CK).

3. Korzystając z obiektów `meuse164` 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.
6. Dodatkowe: stwórz estymację zmiennej `savi` z obiektu `punkty` używając krigingu zwykłego. Porównaj uzyskaną estymację z estymacją przedstawioną w tym rozdziale, stworzoną używając kokrigingu.

# 11. Estymacja lokalnego rozkładu prawdopodobieństwa

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

```
library(sf)
library(stars)
library(gstat)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 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.
##   7.883  11.953  14.937  15.223  17.584  24.945
```

```
punkty$temp_ind = punkty$temp > 20
summary(punkty$temp_ind)
```

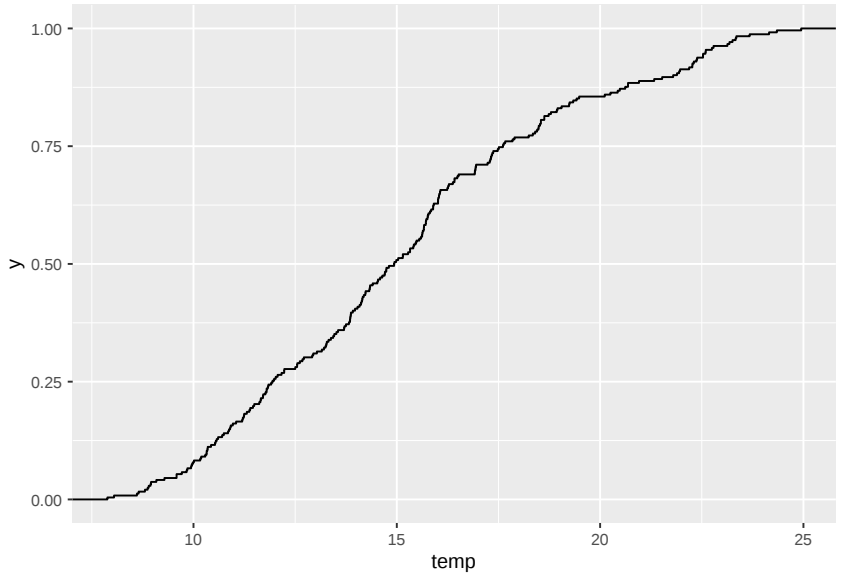
```
##    Mode   FALSE    TRUE
## logical     207      35
```

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 (rycina 11.1).

```
ggplot(punkty, 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 (ryciny 11.2 i 11.3). Korzystając z funkcji `variogram()` tworzony jest semiwariogram empiryczny, używając `vgm()` tworzony jest model "ręczny", który następnie jest dopasowywany z użyciem funkcji `fit.variogram()`.




Rycina 11.1.: Dystrybuanta zmiennej temp.

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

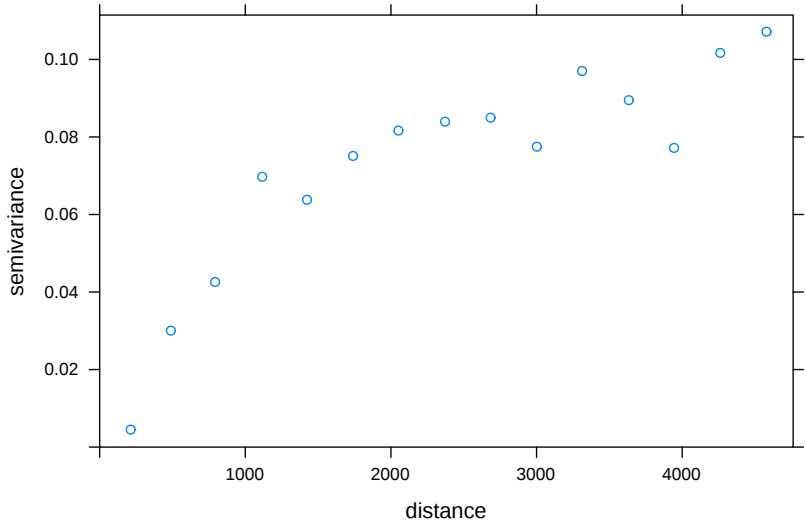
```
model_ind = vgm(model = "Sph", nugget = 0.01)
fitted_ind = fit.variogram(vario_ind, model_ind)
fitted_ind
```

```
##   model      psill    range
## 1   Nug 0.00000000    0.000
## 2   Sph 0.08703823 2342.636
```

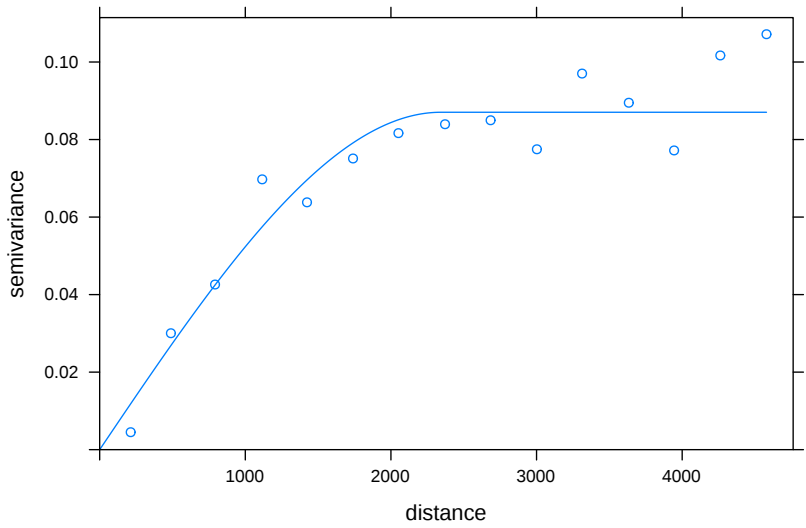
```
plot(vario_ind, model = fitted_ind)
```

### 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`).




Rycina 11.2.: Semiwariogram empiryczny binarnej zmiennej.




Rycina 11.3.: Model semiwariogramu empirycznego binarnej zmiennej.

```
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 (rycina 11.4).

```
plot(ik["var1.pred"], col = paleta)
plot(ik["var1.var"], col = paleta)
```

### 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 (rycina 11.5).

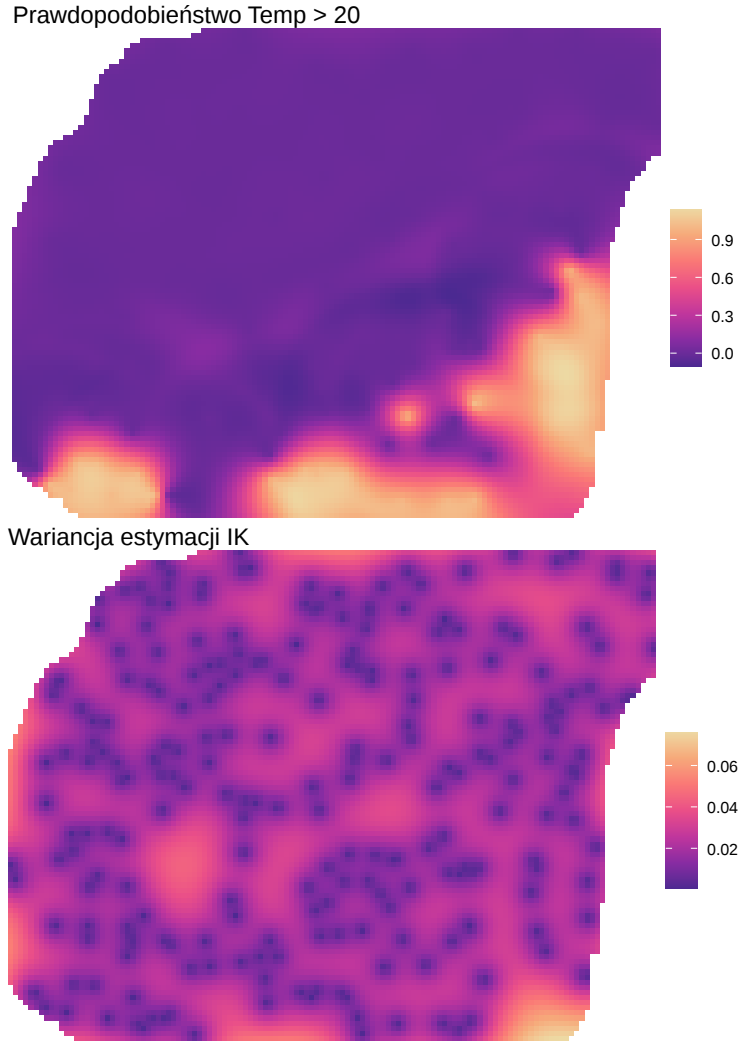
```
ik_df = as.data.frame(ik)
ggplot(ik_df, aes(var1.pred)) + geom_histogram()
```

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.

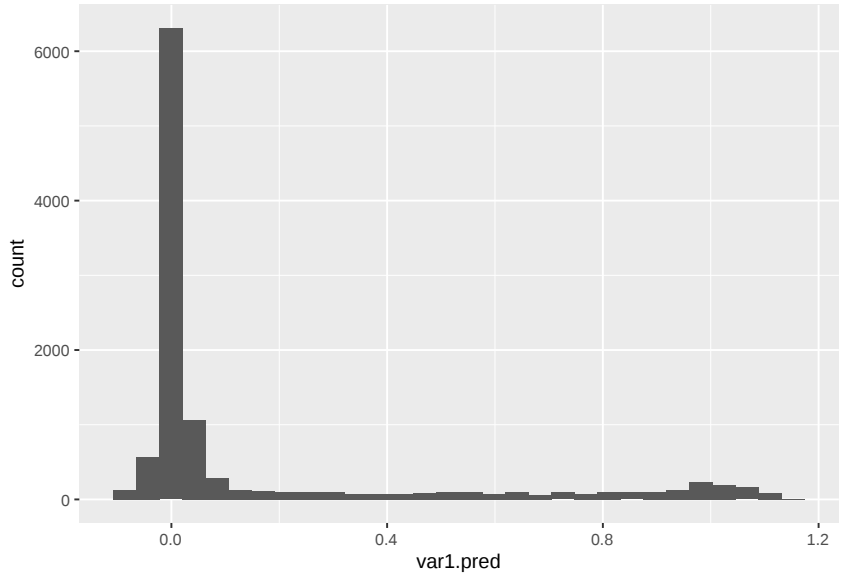
```
ik$prog1 = ik$var1.pred > 0.1
ik$prog2 = ik$var1.pred > 0.75
```

W wyniku otrzymuje się binarne mapy, dla których stwierdza się obszary powyżej lub poniższej zadanego prawdopodobieństwa (rycina 11.6).

```
plot(ik["prog1"], col = c("#88CCEE", "#CC6677"))
plot(ik["prog2"], col = c("#88CCEE", "#CC6677"))
```




Rycina 11.4.: Estymacja i wariancja estymacji używając metody krigingu danych kodowanych (IK).



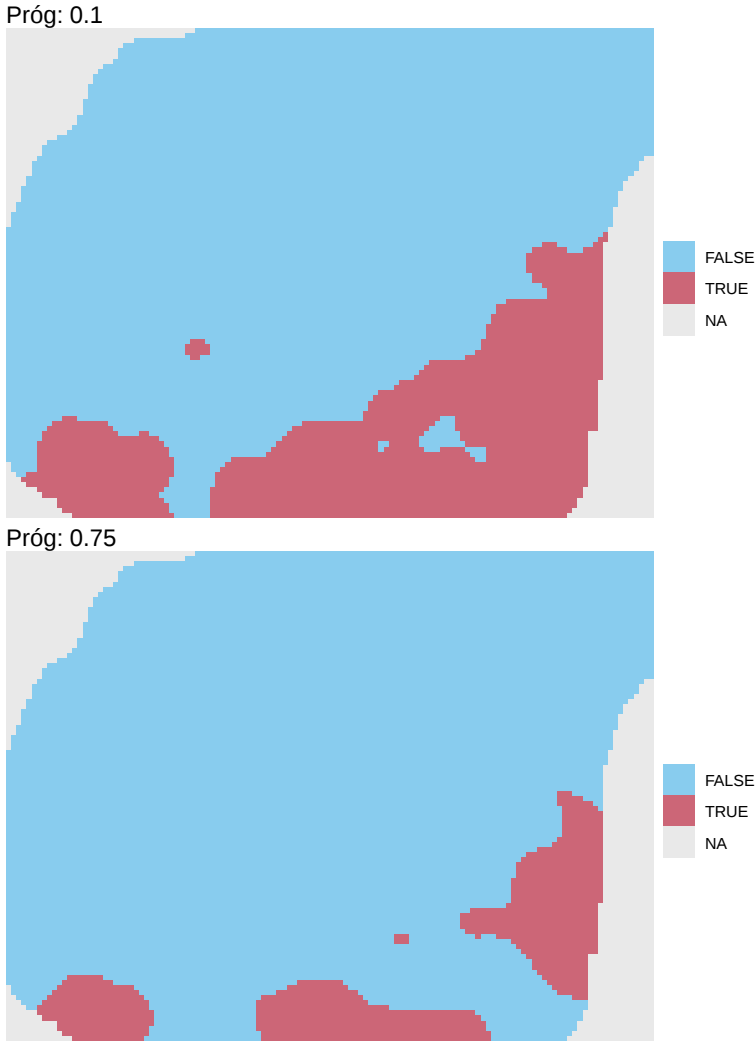

Rycina 11.5.: Rozkład wartości estymacji używając metody krigingu danych kodowanych (IK).

### 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 (rycina 11.7).

```
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,
```




Rycina 11.6.: Binarne mapy dla progu ustalonego na 0.1 oraz 0.75.

```
              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)
```

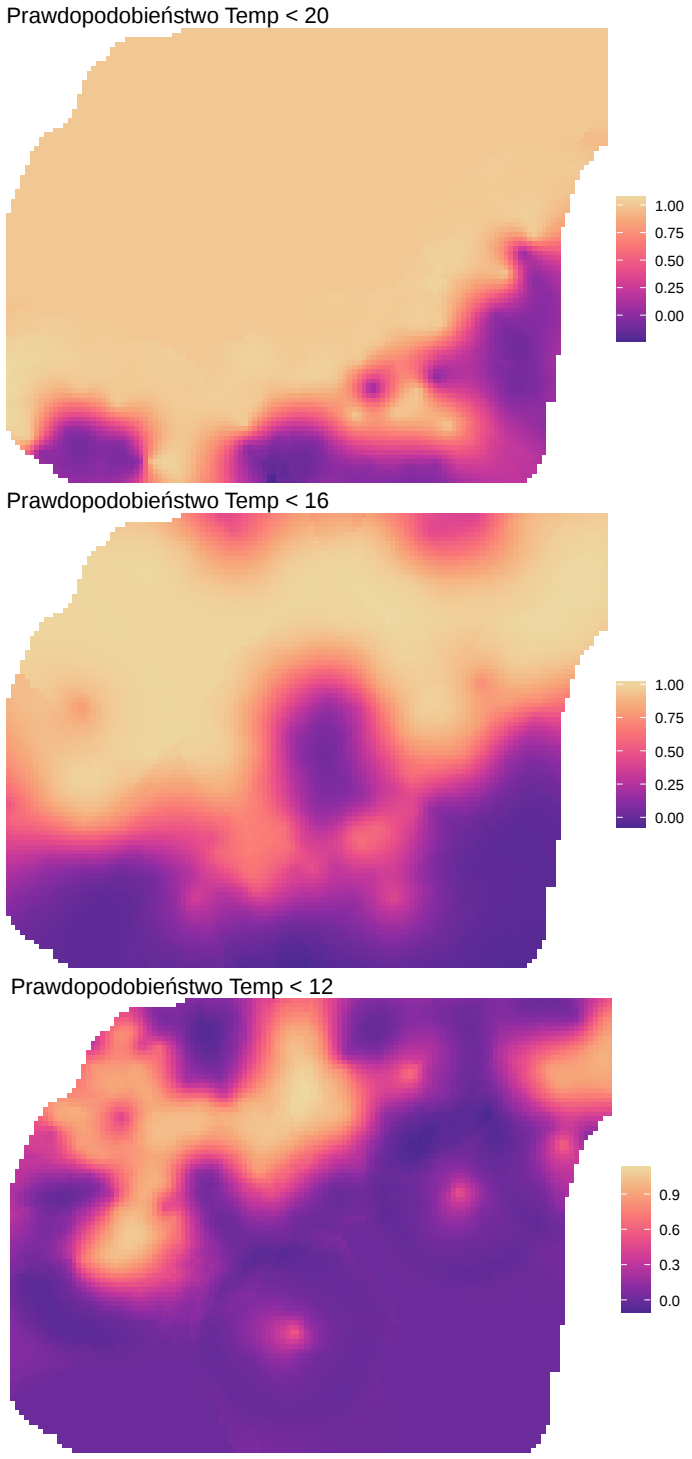
```
## [using ordinary kriging]
```

## 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.




Rycina 11.7.: Estymacje używając metody krigingu danych kodowanych (IK) dla różnych progów prawdopodobieństwa.

# 12. Ocena jakości estymacji

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

```
library(sf)
library(stars)
library(gstat)
library(rsample)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 12.1. Wizualizacja jakości estymacji

### 12.1.1. Mapa

Do oceny przestrzennej jakości estymacji można zastosować mapę przestawiającą błędy estymacji (rycina 12.1). Wyliczenie błędów estymacji odbywa się poprzez odjęcie od estymacji wartości obserwowanej.

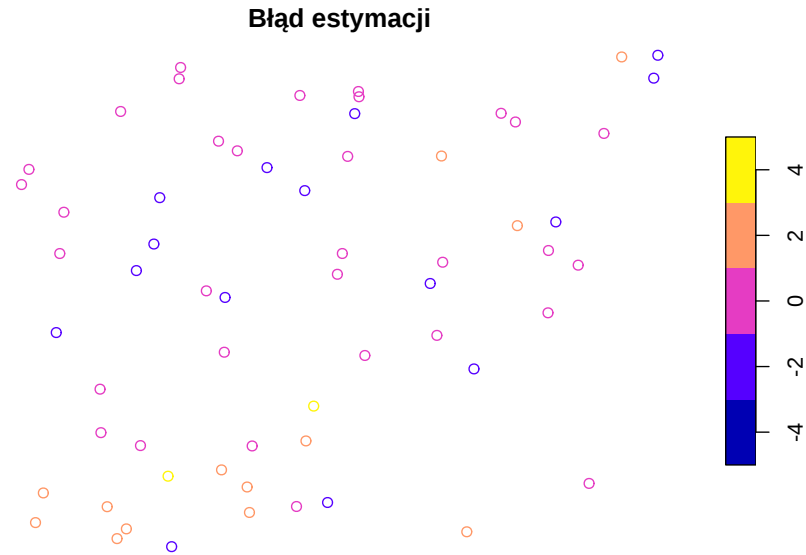
```
blad_estymacji = obserwowane - estymacja
```

### 12.1.2. Histogram

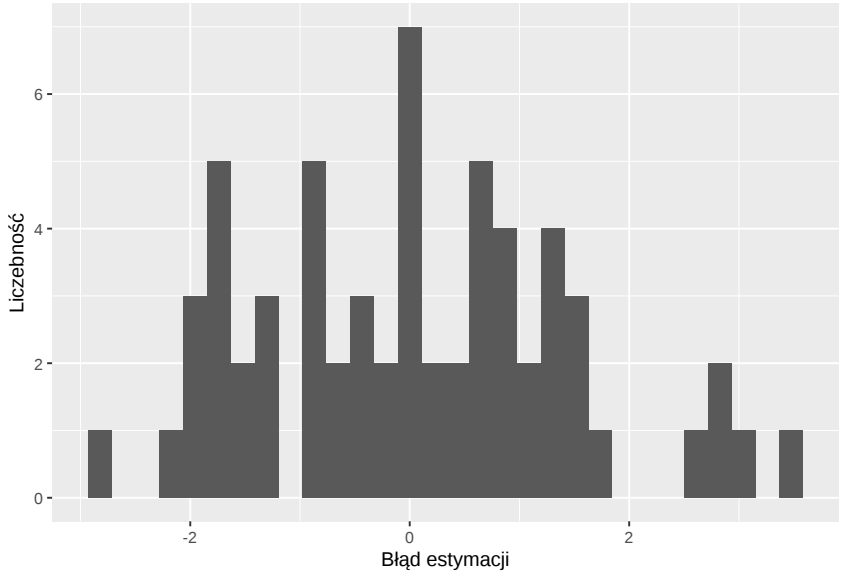
Błąd estymacji można również przedstawić na wykresach, między innymi, na histogramie (rycina 12.2).

### 12.1.3. Wykres rozrzutu

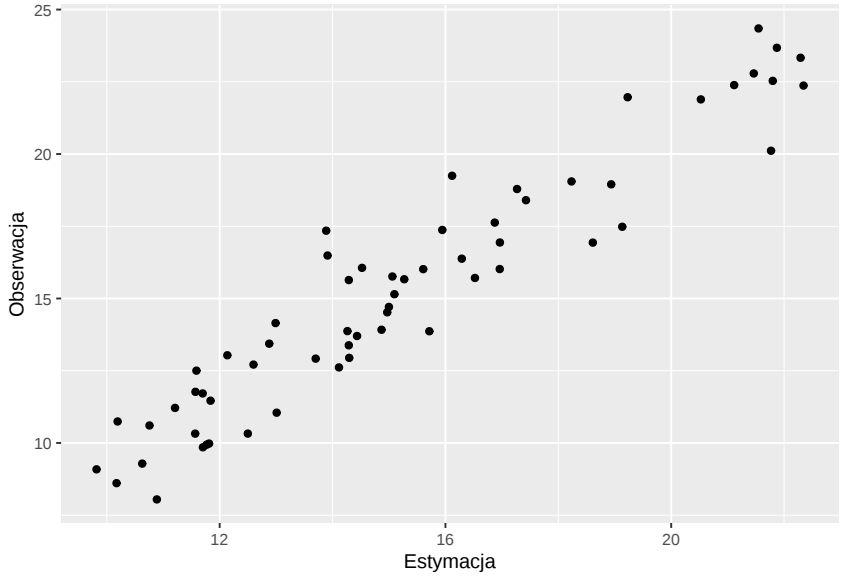
Do porównania pomiędzy wartością estymowaną a obserwowaną może również posłużyć wykres rozrzutu (rycina 12.3).




Rycina 12.1.: Mapa przedstawiająca rozkład przestrzenny błędów estymacji.




Rycina 12.2.: Rozkład wartości błędów estymacji.




Rycina 12.3.: Wykres rozrzutu porównujący rzeczywiste (obserwowane) wartości i wartości estymowane.

## 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 błędu średniokwadratowego (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 błędu średniokwadratowego

Pierwiastek błędu średniokwadratowego (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 błędu średniokwadratowego 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.

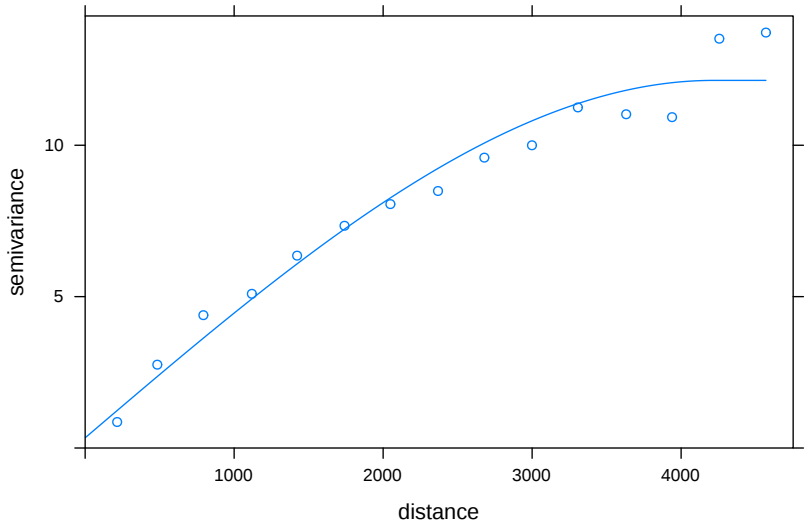
Na poniższym przykładzie zbiór danych dzielony jest używając funkcji `initial_split()` z pakietu **rsample**. Użycie tej funkcji powoduje obiektu zawierającego wydzielony podzbiór treningowy i testowy. Ważną zaletą funkcji `initial_split()` jest to, iż w zbiorze treningowym i testowym zachowane są podobne rozkłady wartości. W przykładzie użyto argumentu

`prop = 0.75`, który oznacza, że 75% danych będzie należało do zbioru treningowego, a 25% do zbioru testowego. Następnie, korzystając ze stworzonego obiektu, budowane są dwa zbiory danych - treningowy (`train`) oraz testowy (`test`).

```
set.seed(224)
punkty_podzial = initial_split(punkty, prop = 0.75, strata = temp)
train = training(punkty_podzial)
test = testing(punkty_podzial)
```

Dalszym krokiem jest stworzenie semiwariogramu empirycznego oraz jego modelowanie w oparciu o zbiór treningowy (rycina 12.4).

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




Rycina 12.4.: Model semiwariogramu w oparciu o zbiór testowy.

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 = 15)
```

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

```
summary(test_sk)
```

```
##    var1.pred          var1.var               geometry
##  Min.   : 9.818   Min.   :0.8199   POINT        :62
##  1st Qu.:12.226   1st Qu.:1.5630   epsg:2180    : 0
##  Median :14.695   Median :2.1911   +proj=tmer...: 0
##  Mean   :15.255   Mean   :2.2243
##  3rd Qu.:17.193   3rd Qu.:2.6707
##  Max.   :22.345   Max.   :5.3092
```

Uzyskane w ten sposób wyniki możemy określić używając statystyk jakości estymacji lub też wykresów (ryciny 12.5, 12.6, 12.7).

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

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.84080 -0.94748  0.02028  0.04646  0.96024  3.46009
```

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

```
## [1] 0.04645957
```

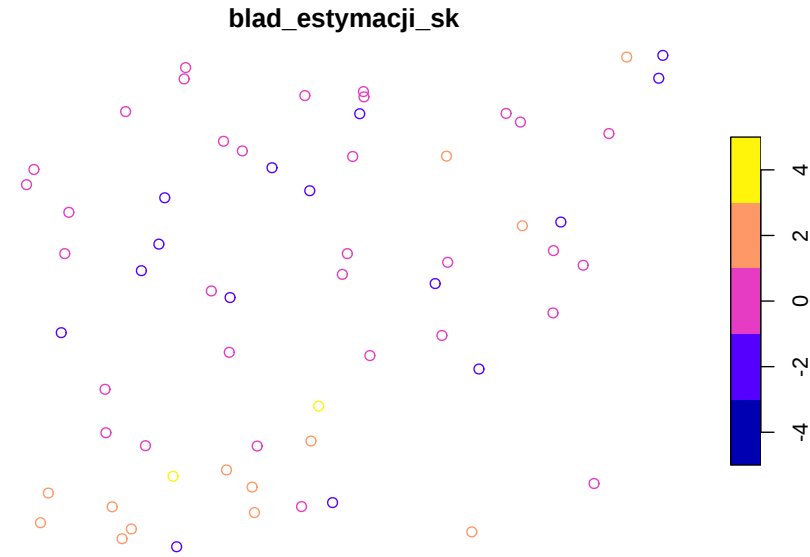
```
RMSE = sqrt(mean((test$temp - test_sk$var1.pred) ^ 2))
RMSE
```

```
## [1] 1.408219
```

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

```
## [1] 0.9038962
```

```
test_sk$blad_estymacji_sk = blad_estymacji_sk
plot(test_sk["blad_estymacji_sk"], breaks = c(-5, -3, -1, 1, 3, 5))
```



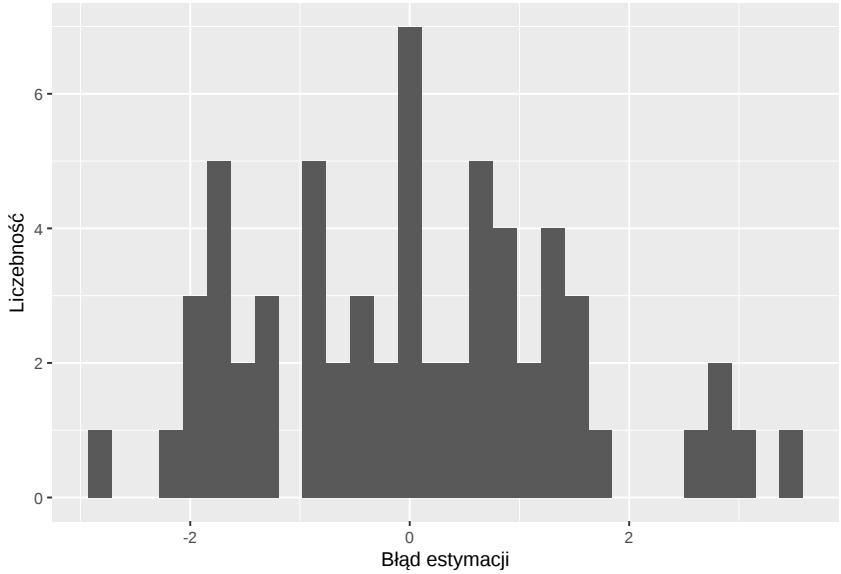

Rycina 12.5.: Mapa przedstawiająca rozkład przestrzenny błędów estymacji uzyskanych używając krigingu prostego.

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

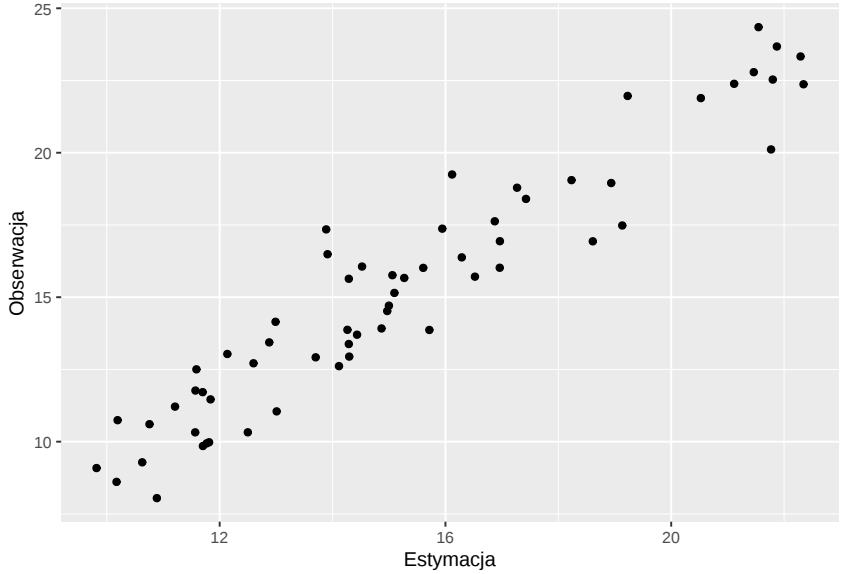
```
test_sk$true = test$temp
ggplot(test_sk, aes(var1.pred, true)) +
    geom_point() +
    xlab("Estymacja") +
    ylab("Obserwacja")
```

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 (rycina 12.8).

```
test_sk = krige(temp ~ 1,
                locations = train,
```




Rycina 12.6.: Rozkład wartości błędów estymacji uzyskanych używając krigingu prostego.

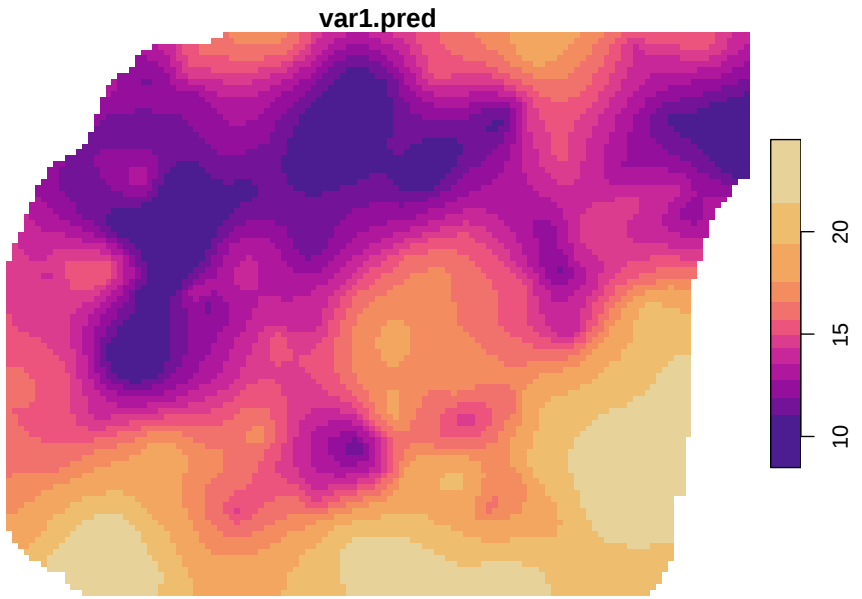



Rycina 12.7.: Wykres rozrzutu porównujący rzeczywiste (obserwowane) wartości i wartości estymowane uzyskane używając krigingu prostego.

```
                newdata = siatka,
                model = fitted,
                beta = 15)
```

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

```
plot(test_sk["var1.pred"], col = paleta)
```




Rycina 12.8.: Estymacja uzyskana dla całego obszaru.

### 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 = 15)
summary(cv_sk)
```

```
##    var1.pred         var1.var        observed         residual
##  Min.   : 8.945   Min.   :1.154   Min.   : 7.883   Min.   :-5.08278
##  1st Qu.:12.258   1st Qu.:1.808   1st Qu.:11.953   1st Qu.:-0.90162
##  Median :14.695   Median :2.177   Median :14.937   Median : 0.05113
##  Mean   :15.235   Mean   :2.245   Mean   :15.223   Mean   :-0.01262
##  3rd Qu.:17.422   3rd Qu.:2.564   3rd Qu.:17.584   3rd Qu.: 0.85763
##  Max.   :23.620   Max.   :5.319   Max.   :24.945   Max.   : 4.30996
##      zscore               fold                 geometry
##  Min.   :-3.555383   Min.   :  1.00   POINT        :242
##  1st Qu.:-0.627964   1st Qu.: 61.25   epsg:NA      :  0
##  Median : 0.030835   Median :121.50   +proj=tmer...:  0
##  Mean   : 0.000425   Mean   :121.50
##  3rd Qu.: 0.636819   3rd Qu.:181.75
##  Max.   : 3.140278   Max.   :242.00
```

Uzyskane w ten sposób wyniki możemy określić używając statystyk jakości estymacji lub też wykresów (ryciny 12.9, 12.10, 12.11).

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

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -5.08278 -0.90162  0.05113 -0.01262  0.85763  4.30996
```

```
MPE = mean(cv_sk$residual)
MPE
```

```
## [1] -0.0126244
```

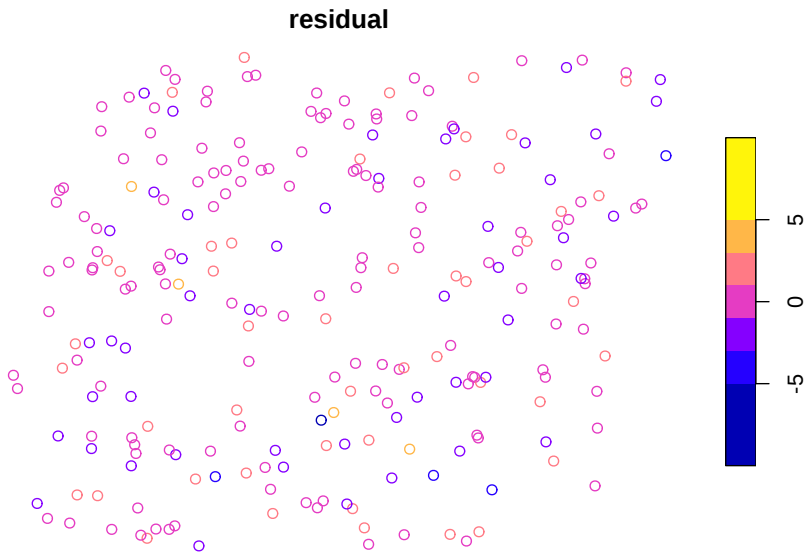
```
RMSE = sqrt(mean((cv_sk$residual) ^ 2))
RMSE
```

```
## [1] 1.40683
```

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

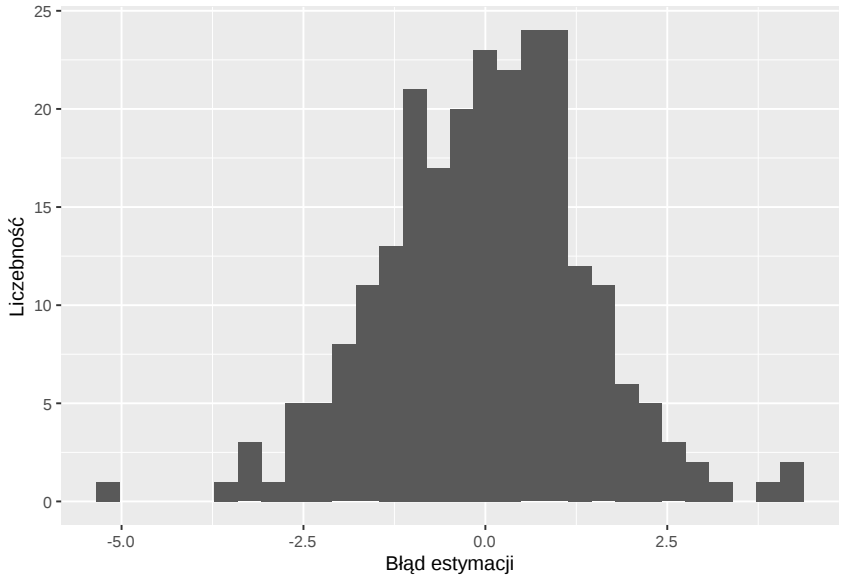
```
## [1] 0.8732558
```

```
plot(cv_sk["residual"], breaks = c(-10, -5, -3, -1, 1, 3, 5, 10))
```




Rycina 12.9.: Mapa przedstawiająca rozkład przestrzenny błędów estymacji uzyskanych używając krigingu prostego i kroswalidacji.

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




Rycina 12.10.: Rozkład wartości błędów estymacji uzyskanych używając krigingu prostego i kroswalidacji.

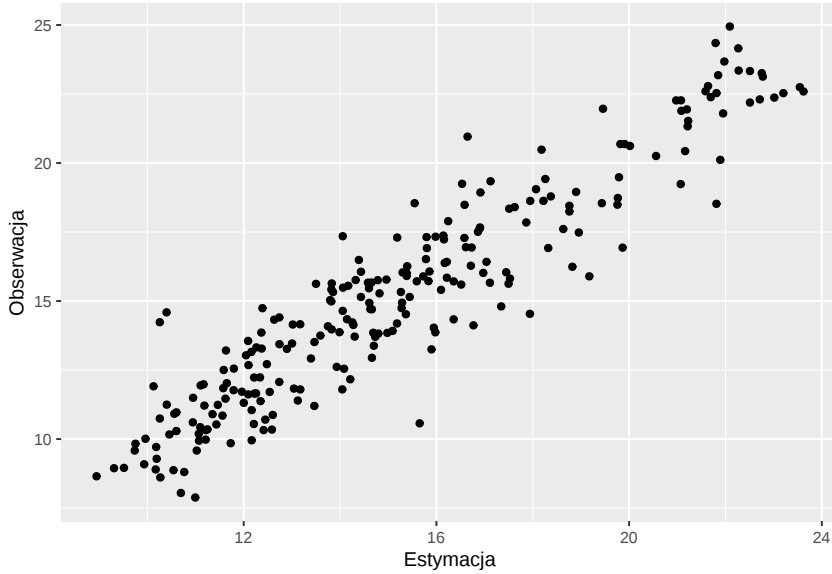
```
ggplot(cv_sk, aes(var1.pred, observed)) +
    geom_point() +
    xlab("Estymacja") +
    ylab("Obserwacja")
```

Podobnie jak w walidacji podzbiorem, gdy uzyskany model jest wystarczająco dobry, estymację dla całego obszaru uzyskuje się z użyciem funkcji `krige()` (rycina 12.12).

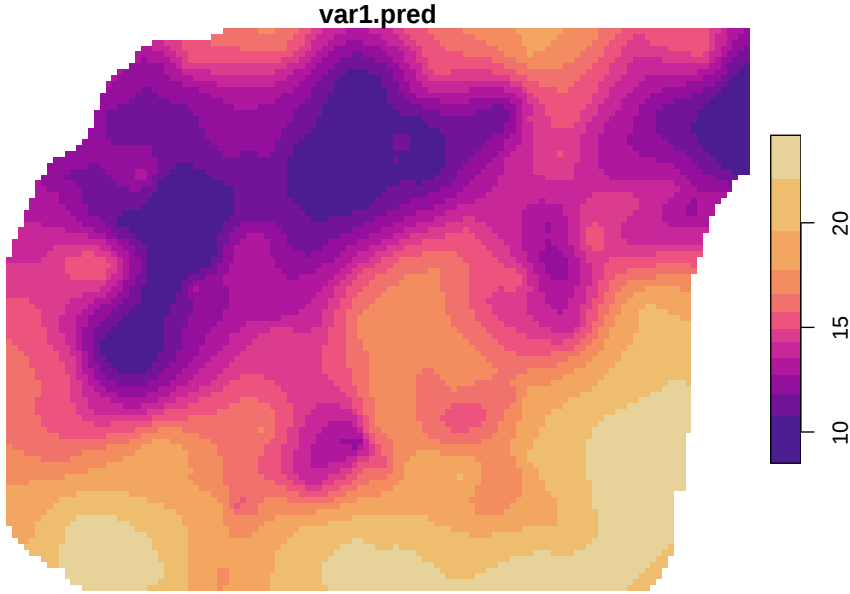
```
cv_skk = krige(temp ~ 1,
                locations = punkty,
                newdata = siatka,
                model = fitted,
                beta = 15)
```

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

```
plot(cv_skk["var1.pred"], col = paleta)
```




Rycina 12.11.: Wykres rozrzutu porównujący rzeczywiste (obserwowane) wartości i wartości estymowane uzyskane używając krigingu prostego i kroswalidacji.




Rycina 12.12.: Estymacja uzyskana dla całego obszaru po oceneniu modelu używając kroswalidacji.

## 12.4. Zadania

1. Wydziel obiekt `punkty` w taki sposób aby 70% danych należało zbioru treningowego, 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?
5. Stwórz optymalne modele zmiennej `temp` ze zbioru treningowego używając krigingu zwykłego, kokrigingu oraz krigingu uniwersalnego, ale tym razem wcześniej transformując wartości tej zmiennej używając metod opisanych w sekcji 3.6. 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? Czy któryś z uzyskanych modeli dla zmiennej po transformacji jest lepszy niż dla zmiennej przed transformacją?

# 13. Symulacje

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

```
library(sf)
library(stars)
library(gstat)
library(ggplot2)
library(geostatbook)
data(punkty)
data(siatka)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

## 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 podstwowe 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

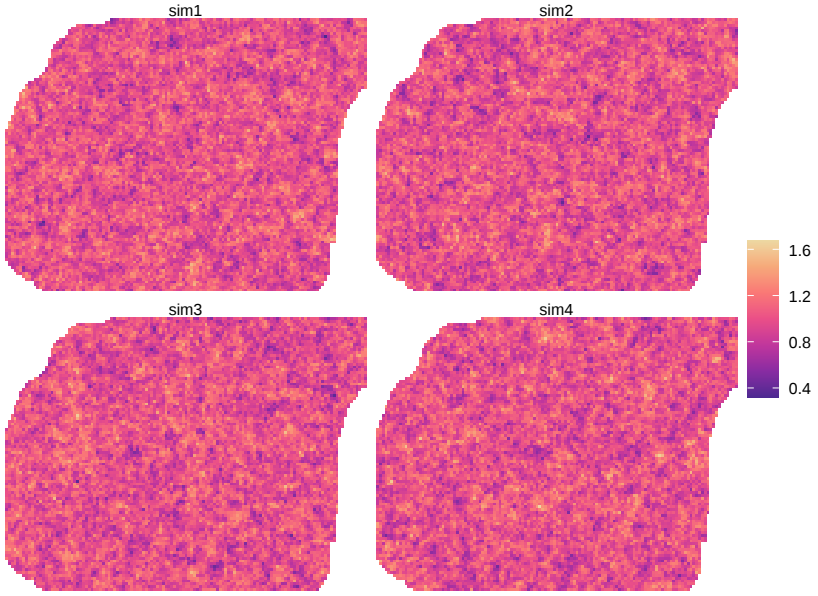
Symulacje bezwarunkowe w pakiecie **gstat** tworzy się z wykorzystaniem funkcji `krige()`. Podobnie jak w przypadku estymacji geostatystycznych, należy tutaj podać wzór, model, siatkę, średnią globalną (`beta`), oraz liczbę sąsiednich punktów używanych do symulacji (w przykładzie poniżej `nmax = 30`). 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 (Ryciny 13.1, 13.2).

```
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]
```

```
plot(sym_bezw1, col = paleta)
```

```
sym_bezw2 = krige(formula = z ~ 1,
                  model = vgm(psill = 0.025,
                              model = "Exp",
                              range = 1500),
                  newdata = siatka,
```




Rycina 13.1.: Przestrzennie skorelowana powierzchnia o średniej równej 1, wartości nsill równej 0.025, zasięgu równym 100, oraz modelu wykładniczego

```
                beta = 1,
                nmax = 30,
                locations = NULL,
                dummy = TRUE,
                nsim = 4)
```

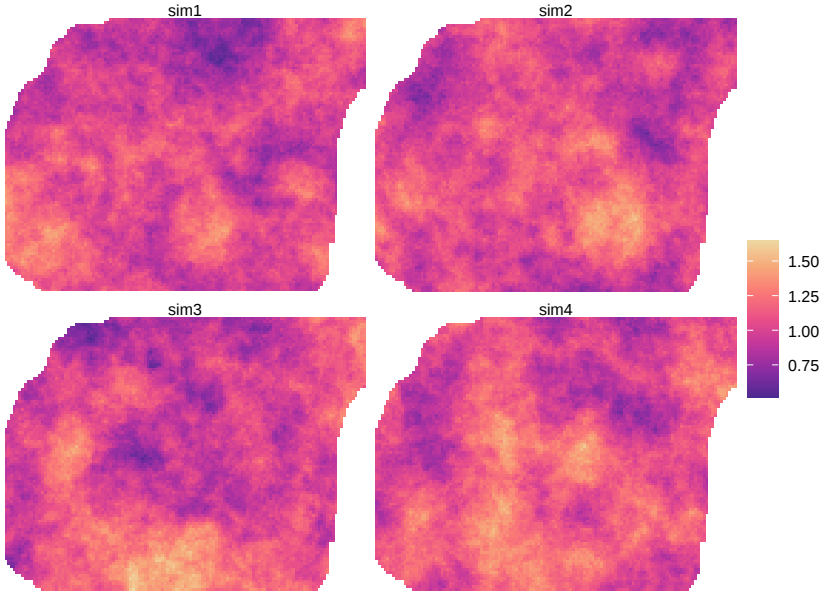
```
## [using unconditional Gaussian simulation]
```

```
plot(sym_bezw2, col = paleta)
```

## 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 wybieraniu losowo lokalizacji nie posiadającej zmierzonej wartości badanej zmiennej:




Rycina 13.2.: Przestrzennie skorelowana powierzchnia o średniej równej 1, wartości nsill równej 0.025, zasięgu równym 1500, oraz modelu wykładniczego

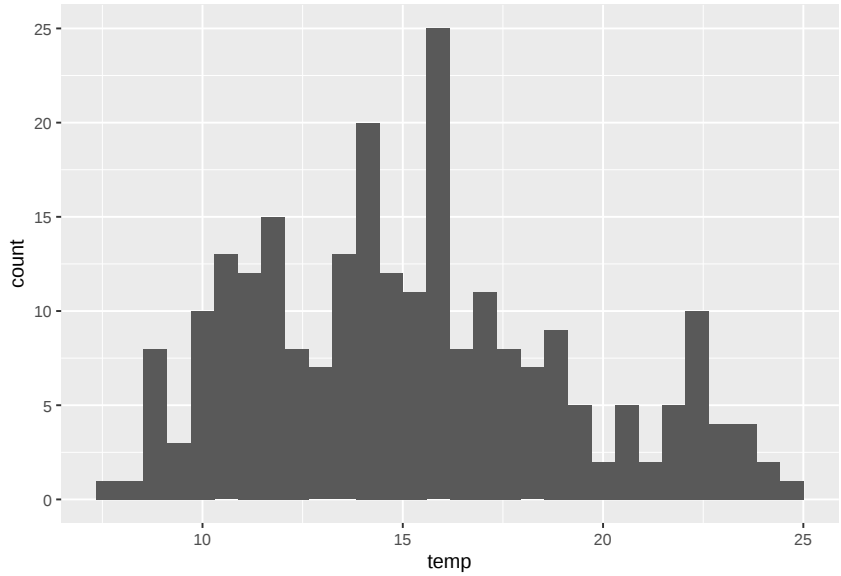
1. Krigingu wartości w tej lokalizacji korzystając z dostępnych danych.
2. Wylosowaniu wartości z rozkładu normalnego o średniej i wariancji wynikającej z krigingu.
3. Dodaniu symulowanej wartości do zbioru danych i przejściu do kolejnej lokalizacji.
4. Powtórzeniu poprzednich kroków, aż do momentu gdy 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) (ryciny 13.3 i 13.4). Zmienna `temp` nie ma rozkładu zbliżonego do normalnego.

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

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

```
##
##  Shapiro-Wilk normality test
```




Rycina 13.3.: Rozkład wartości zmiennej temp.

```
##
## data:  punkty$temp
## W = 0.96904, p-value = 0.00004054
```

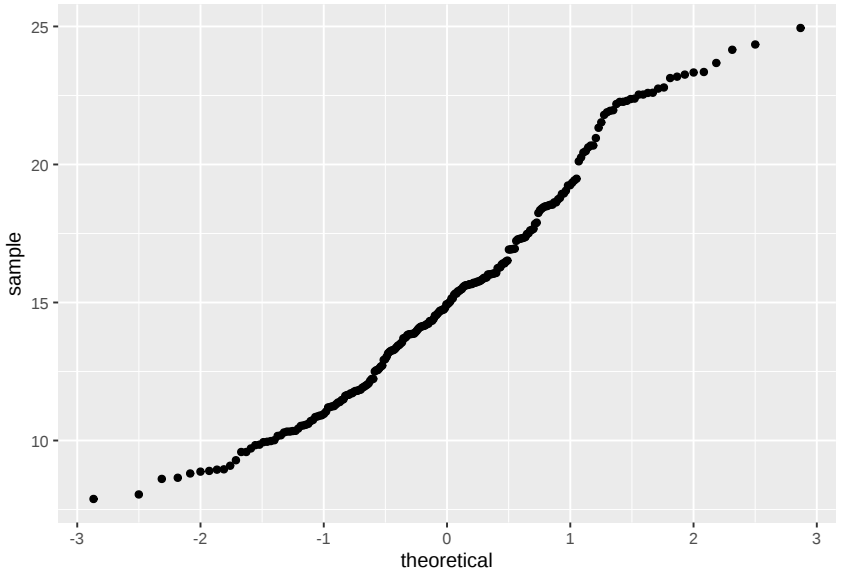
Na potrzeby symulacji zmienna `temp` została zlogarytmizowna (ryciny 13.5 i 13.6).

```
punkty$temp_log = log(punkty$temp)
ggplot(punkty, aes(temp_log)) + geom_histogram()
```

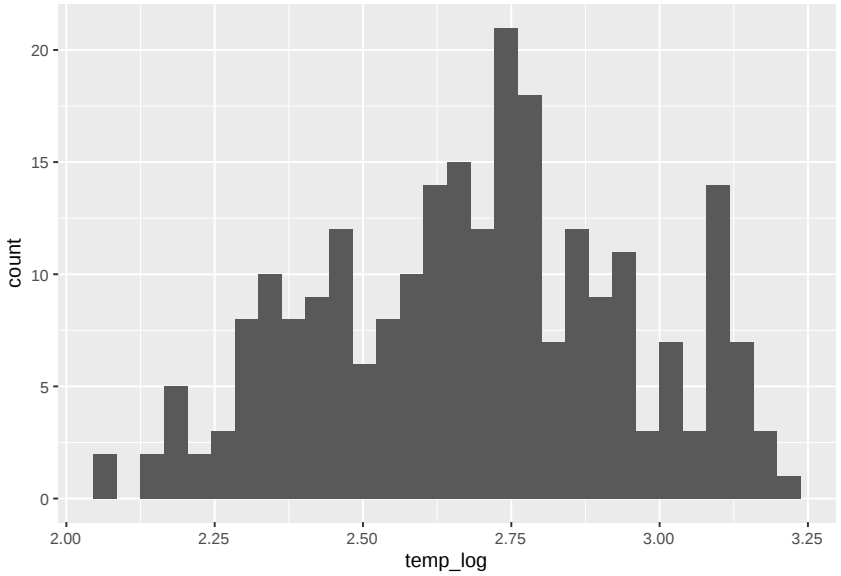
```
ggplot(punkty, aes(sample = temp_log)) + stat_qq()
shapiro.test(punkty$temp_log)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  punkty$temp_log
## W = 0.98435, p-value = 0.009226
```

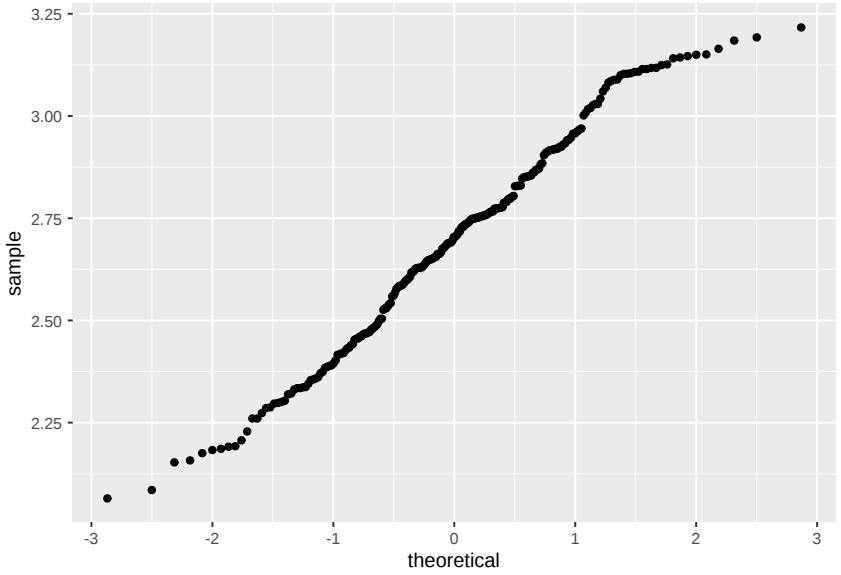
Dalsze etapy przypominają przeprowadzenie estymacji statystycznej, jedynym wyjątkiem jest dodanie argumentu mówiącego o liczbie symulacji do przeprowadzenia (`nsim` w funkcji `krige()`) (ryciny 13.7 i 13.8).




Rycina 13.4.: Wykres kwantyl-kwantyl wartości zmiennej temp.




Rycina 13.5.: Rozkład wartości zmiennej temp po logaritmizacji.




Rycina 13.6.: Wykres kwantyl-kwantyl wartości zmiennej temp po logarytmizacji.

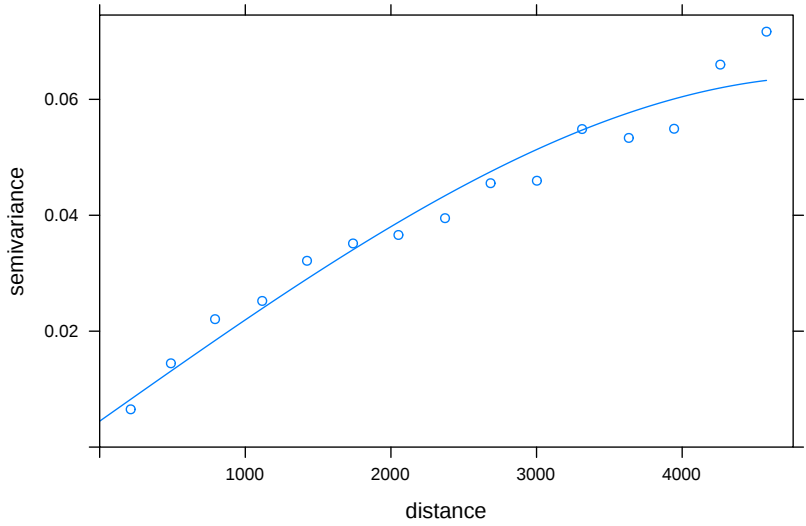
```
vario = variogram(temp_log ~ 1, locations = punkty)
model = vgm(model = "Sph", nugget = 0.005)
fitted = fit.variogram(vario, model)
plot(vario, model = fitted)
```

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

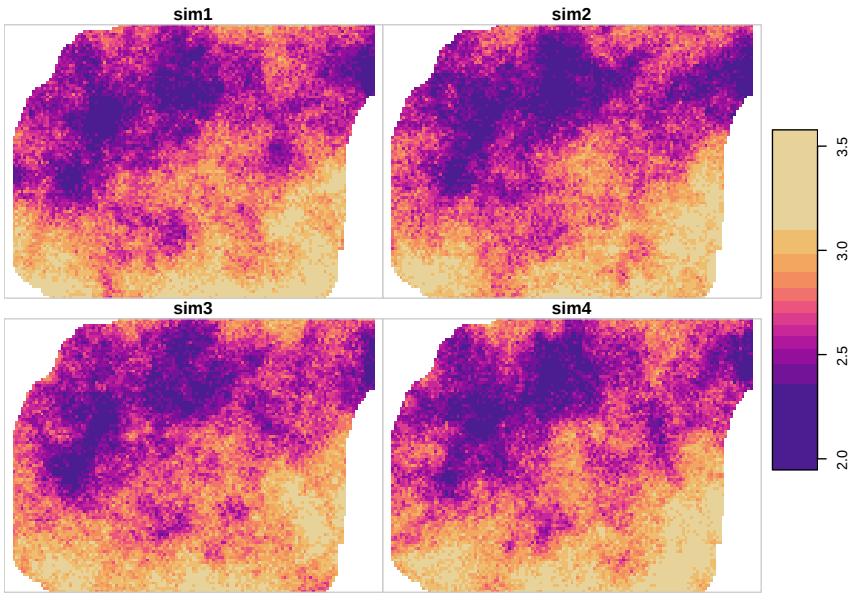
```
## drawing 4 GLS realisations of beta...
## [using conditional Gaussian simulation]
```

```
plot(sym_ok, col = paleta)
```

```
## downsample set to c(0,0,1)
```



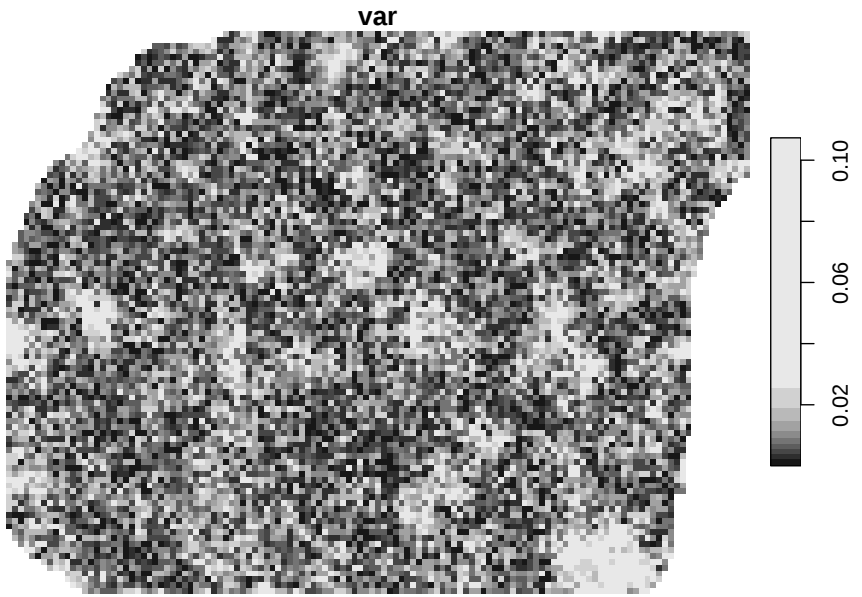

Rycina 13.7.: Model semiwariogramu zmiennej temp po logaritmizacji




Rycina 13.8.: Symulacja warunkowa zmiennej temp po logaritmizacji.

Wyniki symulacji można przetworzyć do pierwotnej jednostki z użyciem funkcji `rev_trans()`, którą stworzyliśmy w sekcji 3.6.4. Potrzebuje ona jednak najpierw określenia wariancji symulacji (rycina 13.9). Do jej wyliczenia używamy kombinacji funkcji `st_apply()` oraz `var()`. W funkcji `st_apply()` konieczne tutaj jest określenie argumentu `MARGIN = c("x", "y")`, co oznacza że funkcja `var` będzie wykonana dla każdej komórki niezależnie. W efekcie tej operacji otrzymujemy tylko jedną warstwę.

```
sym_ok_var = st_apply(sym_ok, MARGIN = c("x", "y"), FUN = var)
plot(sym_ok_var)
```



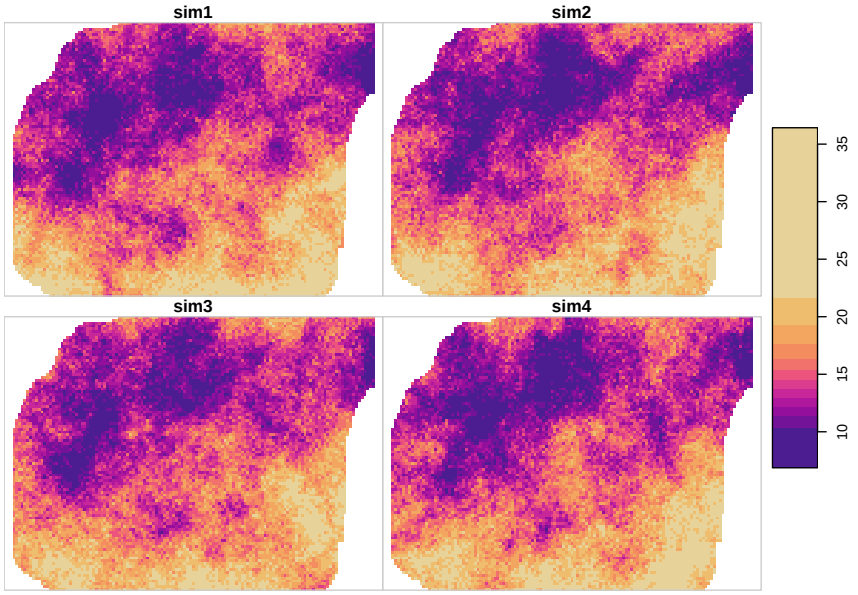

Rycina 13.9.: Wariancja symulacji warunkowej.

Teraz możliwe jest przywrócenie oryginalnej jednostki dla wszystkich symulacji z użyciem funkcji `st_apply()` i `rev_trans()`. (rycina 13.10)

```
sym_ok_rescaled = st_apply(sym_ok, 3, rev_trans,
                           var = sym_ok_var$var, obs = punkty$temp)
plot(sym_ok_rescaled, col = paleta)
```

```
## downsample set to c(0,0,1)
```

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




Rycina 13.10.: Symulacja warunkowa po przeskalowaniu do oryginalnych wartości.

```
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 trzykrotnie z funkcji `st_apply()`. Przywrócenie oryginalnej jednostki odbywa się poprzez wyliczenie wariancji symulacji, a następnie użycie funkcji `rev_trans()`.

```
# wyliczenie wariancji (jedostka log)
sym_sk_var = st_apply(sym_sk, MARGIN = c("x", "y"), FUN = var)
# przywrócenie oryginalnej jednostki
sym_sk_rescaled = st_apply(sym_sk, 3, rev_trans,
                           var = sym_sk_var$var, obs = punkty$temp)
sym_sk_rescaled
```

```
## stars object with 3 dimensions and 1 attribute
## attribute(s), summary of first 100000 cells:
##           Min.  1st Qu.   Median    Mean  3rd Qu.    Max.  NA's
## var1  6.213417 12.19311 14.54702 15.1666 17.55568 34.2181 10550
## dimension(s):
##        from  to offset delta               refsys point          values x/y
## x         1 127 745542    90 ETRS89 / Poland CS92    NA            NULL [x]
## y         1  96 721256   -90 ETRS89 / Poland CS92    NA            NULL [y]
## sample    1 100     NA    NA                   NA    NA sim1,...,sim100
```

Wyliczenie odchylenia standardowego odbywa się poprzez argumenty `MARGIN = c("x", "y")` oraz `FUN = sd`. W ten sposób funkcja `sd()` będzie wykonana niezależnie dla każdej komórki dla wszystkich warstw. W efekcie tej operacji otrzymuje się tylko jedną warstwę.

```
# wyliczenie odchylenia standardowego
sym_sk_sd = st_apply(sym_sk_rescaled, MARGIN = c("x", "y"), FUN = sd)
sym_sk_sd
```

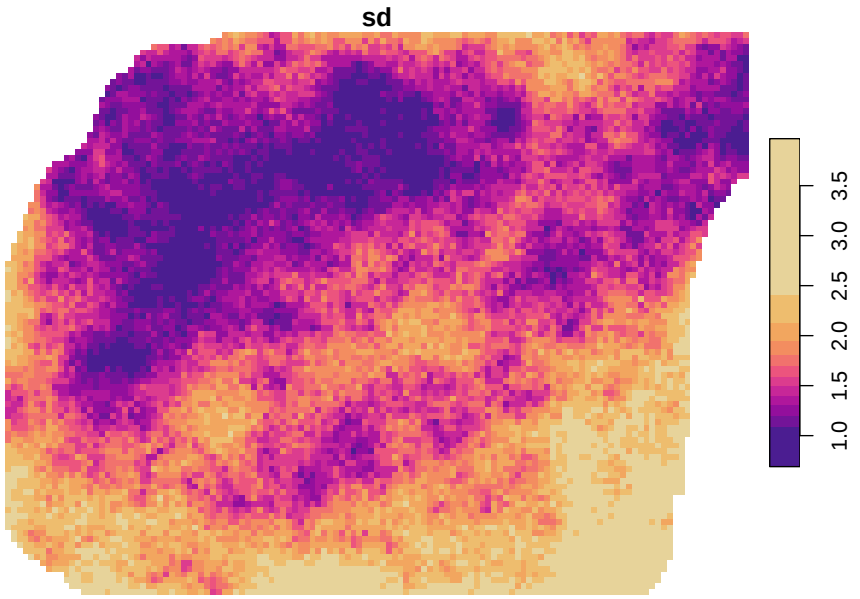
```
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##          Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## sd  0.6898883 1.303663 1.595296 1.669856 1.948577 3.969843 1270
## dimension(s):
##   from  to offset delta               refsys point values x/y
## x    1 127 745542    90 ETRS89 / Poland CS92    NA   NULL [x]
## y    1  96 721256   -90 ETRS89 / Poland CS92    NA   NULL [y]
```

Finalnie otrzymujemy mapę odchylenia standardowego symulowanych wartości (rycina 13.11). Można na niej odczytać obszary o najpewniejszych (najmniej zmiennych) wartościach (niebieski kolor) oraz obszary o największej zmienności cechy (kolor żółty).

```
plot(sym_sk_sd, col = paleta)
```

### 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.




Rycina 13.11.: Mapa odchylenia standardowego symulowanych wartości.

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.883  11.953  14.937  15.223  17.584  24.945
```

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

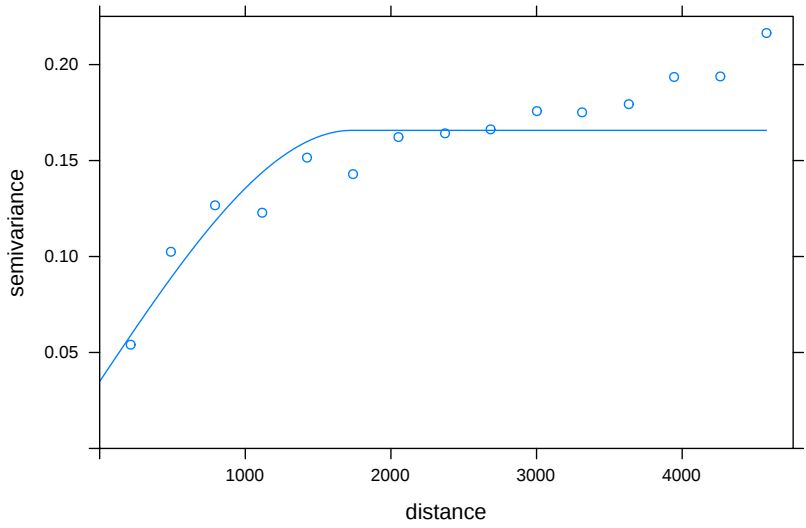
```
##    Mode   FALSE    TRUE
## logical     180      62
```

W tej metodzie kolejne etapy przypominają przeprowadzenie krigingu danych kodowanych (rycina 13.12). 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 0.03492253    0.000
## 2   Sph 0.13081796 1733.339
```

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




Rycina 13.12.: Model semiwariogramu empirycznego binarnej zmiennej stworzony na potrzeby symulacji.

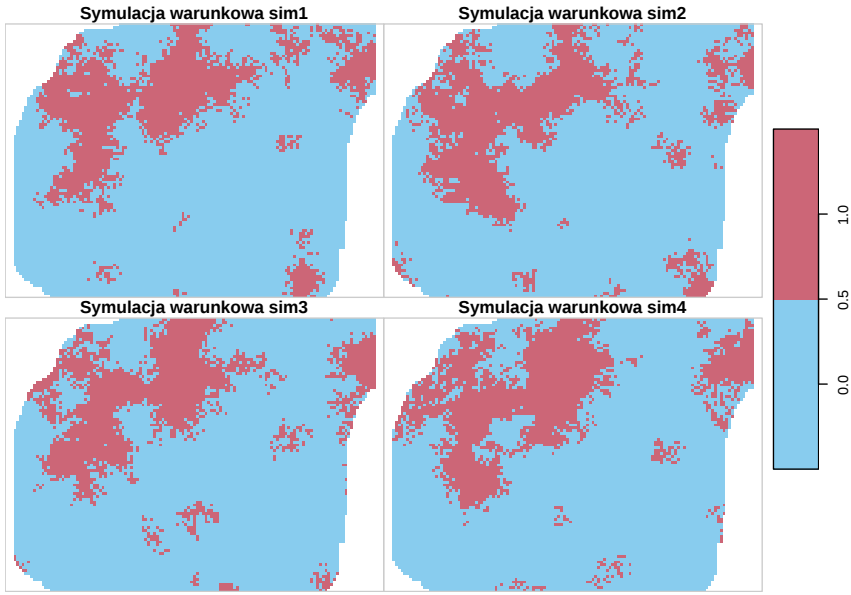
```
sym_ind = krige(temp_ind ~ 1,
                locations = punkty,
                newdata = siatka,
                model = fitted_ind,
                indicators = TRUE,
                nsim = 4,
                nmax = 30)
```

```
## 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` (rycina 13.13).

```
plot(sym_ind, main = "Symulacja warunkowa",
     col = c("#88CCEE", "#CC6677"))
```

```
## downsample set to c(0,0,1)
```




Rycina 13.13.: Przykłady symulacji danych kodowanych.

## 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

- Elementarz programisty: Wstęp do programowania używając R[1] (Nowosad, 2020)
- Przewodnik po pakiecie R[2] (Biecek, 2014)
- Programowanie w języku R Analiza Danych. Obliczenia. Symulacje[3] (Gagolewski, 2016)
- R for Data Science - https://r4ds.had.co.nz/ - https://helion.pl/ksiazki/jezyk-r-kompletny-zestaw-narzedzi-dla-analitykow-danych-hadley-wickham-garrett-grolemund,jezrkv.htm#format/d (Wickham and Grolemund, 2016)
- Efficient R programming - https://csgillespie.github.io/efficientR/ - https://helion.pl/ksiazki/wydajne-programowanie-w-r-praktyczny-przewodnik-po-lepszym-programowaniu-colin-gillespie-robin-lovelace,e_1vi4.htm#format/e (Gillespie and Lovelace, 2016)
- Metody przetwarzania danych meteorologicznych w języku programowania R - http://enwo.pl/przetwarzanie (Czernecki, 2018)

## A.2. Analizy przestrzenne w R

- Geocomputation with R[4] (Lovelace et al., 2019)
- Applied Spatial Dala Analysis with R[5] (Bivand et al., 2014)
- Spatial Data Science[6] (Pebesma and Bivand, 2019)
- CRAN Task View: Analysis of Spatial Data[7]

[1] https://nowosad.github.io/elp/
[2] http://www.biecek.pl/R/
[3] http://www.gagolewski.com/publications/programowanier/
[4] https://geocompr.robinlovelace.net/
[5] http://www.asdar-book.org/
[6] https://www.r-spatial.org/book/
[7] https://cran.r-project.org/web/views/Spatial.html

## A.3. Geostatystyka

- Praktyczny poradnik - jak szybko zrozumieć i wykorzystać geostatystykę w pracy badawczej[8] (Stach, 2009)
- An introduction to geostatistics with R/gstat[9] (Rossiter, 2016)
- A Practical Guide to Geostatistical Mapping[10] (Hengl, 2009)
- gstat user's manual[11] (Pebesma, 2001)
- Applied Geostatistics[12] (Isaaks and Srivastava, 1989)
- Statistics for spatial data[13] (Cressie, 1992)
- Geostatistics for Natural Resources Evaluation[14] (Goovaerts et al., 1997)
- Geostatistics for Environmental Scientists[15] (Webster and Oliver, 2007)

[8] http://www.geoinfo.amu.edu.pl/staff/astach/www_geostat/programy/A_Stach_%20poradnik_geostatystyki.pdf
[9] http://www.css.cornell.edu/faculty/dgr2/_static/files/R_PDF/gs_intro_20Mar2019.pdf
[10] https://library.wur.nl/isric/fulltext/isricu_i27272_001.pdf
[11] http://www.gstat.org/gstat.pdf
[12] https://books.google.pl/books?id=vC2dcXFLI3YC
[13] https://books.google.pl/books?id=4SdRAAAAMAAJ
[14] http://www.oupcanada.com/catalog/9780195115383.html
[15] 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("remotes")
remotes::install_github("nowosad/geostatbook@3")
```

```
library(geostatbook)
```

## B.1. `punkty`

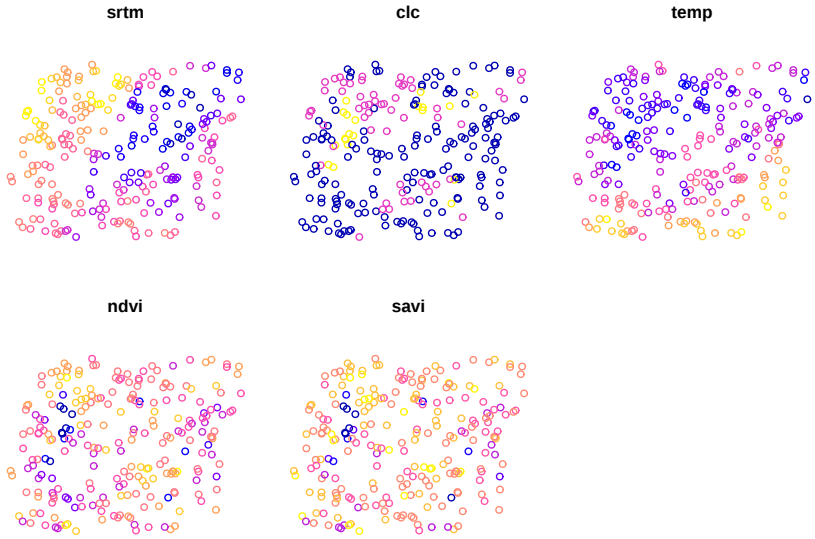
```
data("punkty")
?punkty
```

Zbiór danych `punkty` zawiera 242 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
- `ndvi` - znormalizowany różnicowy wskaźnik wegetacji (ang. *Normalized Difference Vegetation Index*)
- `savi` - wskaźnik wegetacji mniej podatny na wpływ jasności gleby na uzyskane wyniki (ang. *Soil Adjusted Vegetation Index*)

[1] https://github.com/Nowosad/geostat_book/blob/master/dane3.zip?raw=true
[2] https://github.com/Nowosad/geostatbook

```
plot(punkty)
```



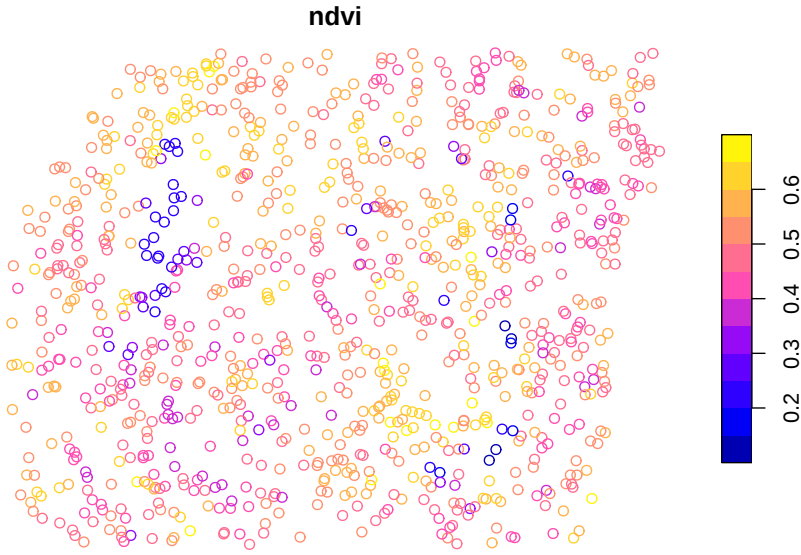

## B.2. `punkty_ndvi`

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

Zbiór danych `punkty_ndvi` zawiera 993 obserwacji oraz 1 zmienną dla obszaru Suwalskiego Parku Krajobrazowego i okolic. Zmienna:

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

```
plot(punkty_ndvi)
```



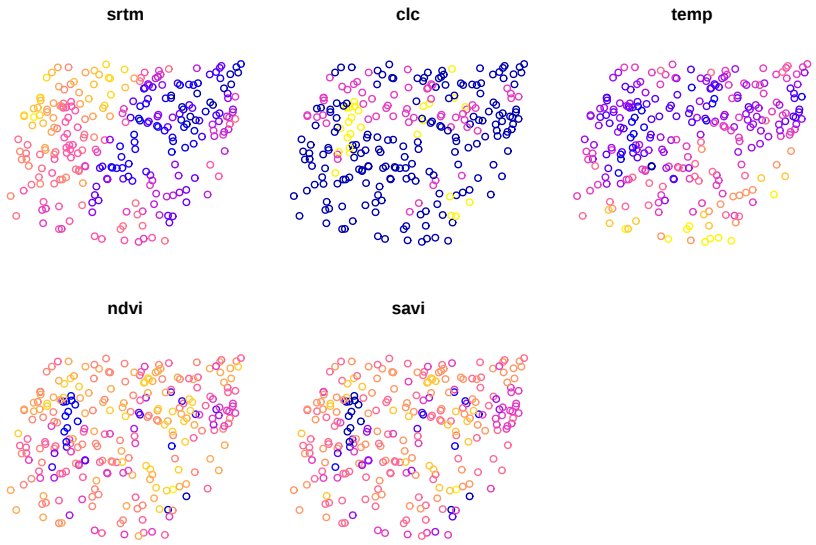

## B.3. `punkty_pref`

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

Zbiór danych `punkty_pref` zawiera 264 obserwacji oraz 5 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` - wskaźnik wegetacji mniej podatny na wpływ jasności gleby na uzyskane wyniki (ang. *Soil Adjusted Vegetation Index*)

```
plot(punkty_pref)
```

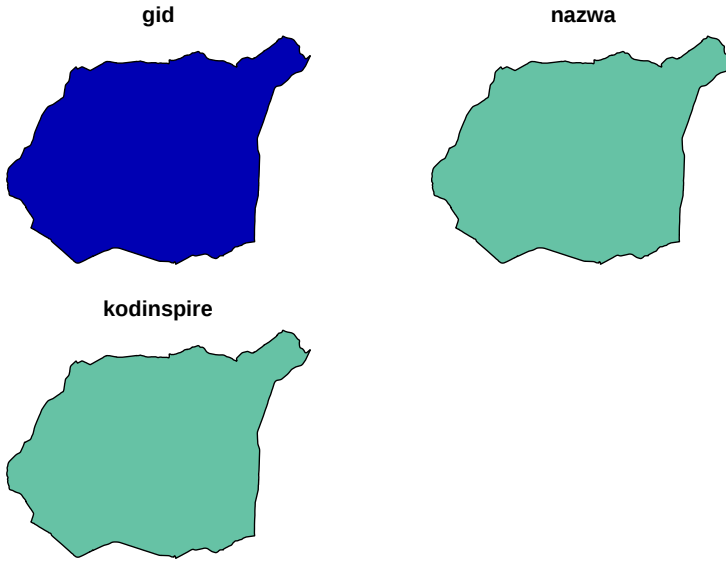



## B.4. granica

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

Granica Suwalskiego Parku Krajobrazowego.

```
plot(granica)
```




## B.5. siatka

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

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

```
plot(siatka)
```


## B.6. dane_uzup

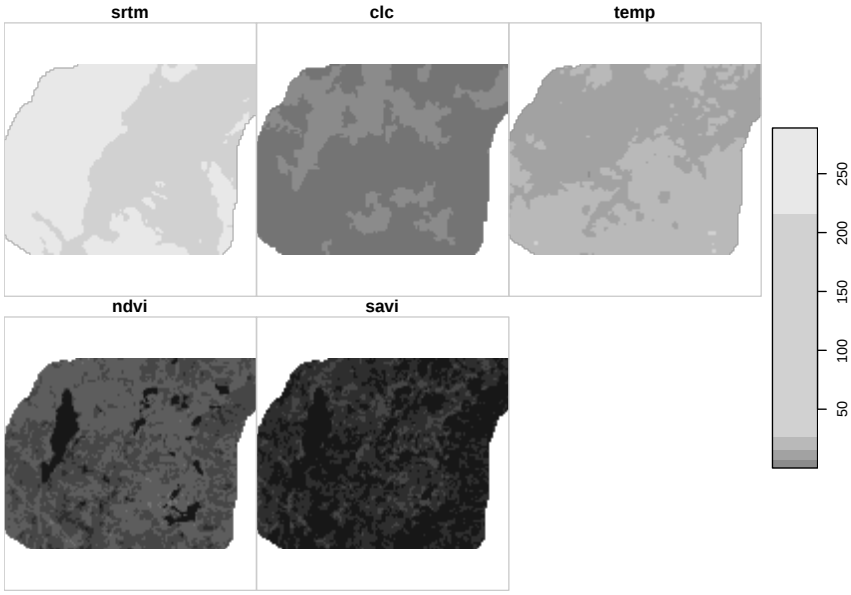
```
data("dane_uzup")
?dane_uzup
```

Siatka badanego obszaru dla obszaru Suwalskiego Parku Krajobrazowego i okolic zawierająca zmienne dodatkowe. Zawiera ona 96 wierszy i 127 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` - wskaźnik wegetacji mniej podatny na wpływ jasności gleby na uzyskane wyniki (ang. *Soil Adjusted Vegetation Index*)

```
plot(st_redimension(dane_uzup))
```

```
## downsample set to c(0,0,1)
```


srtm
clc
temp
ndvi
savi
250
200
150
100
50

# C. Przykład analizy geostatystycznej

## C.1. Analiza geostatystyczna

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

## C.2. Przygotowanie danych

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

```
library(sf)
library(stars)
library(gstat)
library(ggplot2)
paleta = hcl.colors(12, palette = "ag_Sunset")
```

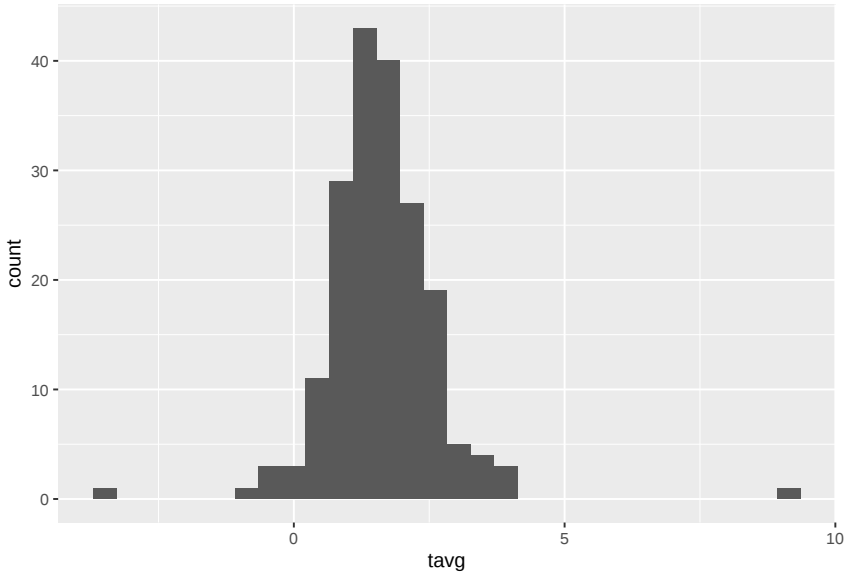
Kolejnym krokiem jest wczytanie danych oraz sprawdzenie ich jakości.

```
moje_punkty = read_sf("dane/moje_punkty.gpkg")
summary(moje_punkty)
```

```
##       tavg             geom
##  Min.   :-3.463   POINT  :190
##  1st Qu.: 1.081   epsg:NA:  0
##  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 (rycina C.1 i C.2).

```
ggplot(moje_punkty, aes(tavg)) + geom_histogram()
```



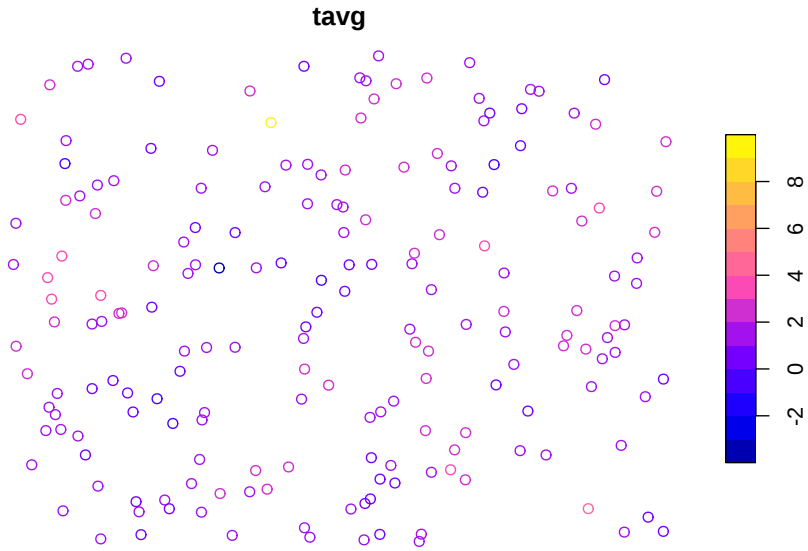

Rycina C.1.: Rozkład wartości zmiennej tavg.

```
plot(moje_punkty)
```

Pozwala to na zauważenie, że w badanej zmiennej występują co najmniej dwie wartości odstające.

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

```
## Simple feature collection with 1 feature and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 349744.4 ymin: 320313.8 xmax: 349744.4 ymax: 320313.8
## CRS:           NA
## # A tibble: 1 x 2
##    tavg                geom
##   <dbl>             <POINT>
## 1  9.16 (349744.4 320313.8)
```




Rycina C.2.: Rozkład przestrzenny wartości zmiennej tavg.

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

```
## Simple feature collection with 1 feature and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 348791.9 ymin: 317639.4 xmax: 348791.9 ymax: 317639.4
## CRS:           NA
## # A tibble: 1 x 2
##    tavg                geom
##   <dbl>             <POINT>
## 1 -3.46 (348791.9 317639.4)
```

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`.

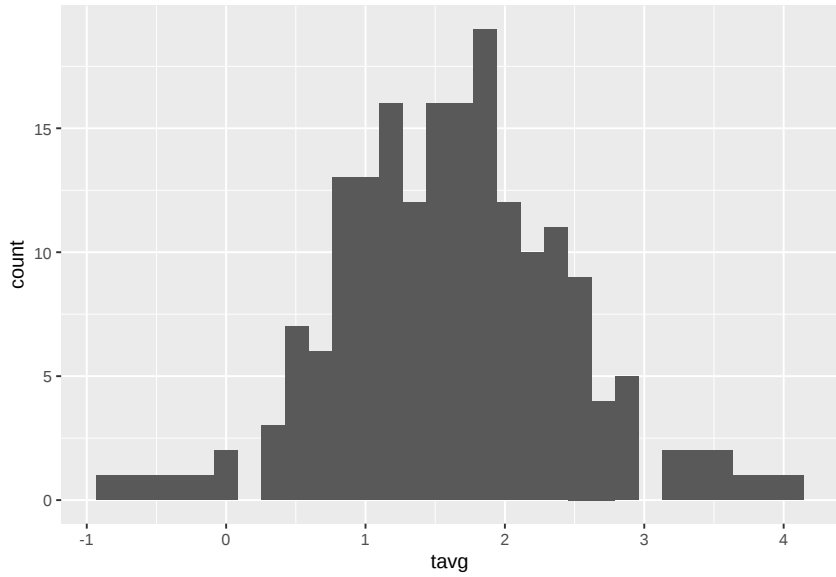
```
# usunięcie wartości według id
moje_punkty[60, "tavg"] = NA
moje_punkty[72, "tavg"] = NA
```

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 (rycina C.3 i C.4).

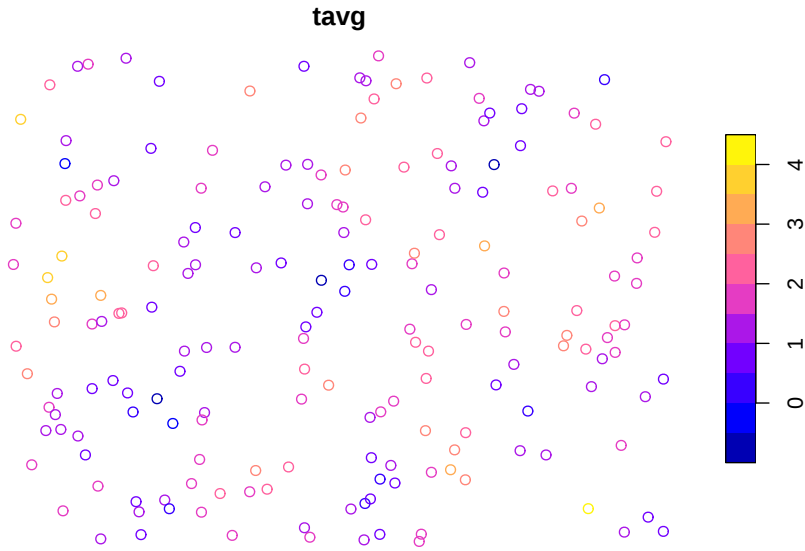
```
ggplot(moje_punkty, aes(tavg)) + geom_histogram()
```




Rycina C.3.: Rozkład wartości zmiennej tavg po usunięciu wartości odstających.

```
plot(moje_punkty)
```

Można dodatkowo stworzyć chmurę semiwariogramu w celu wyszukania potencjalnych wartości lokalnie odstających (rycina C.5).




Rycina C.4.: Rozkład przestrzenny wartości zmiennej tavg po usunięciu wartości odstających.

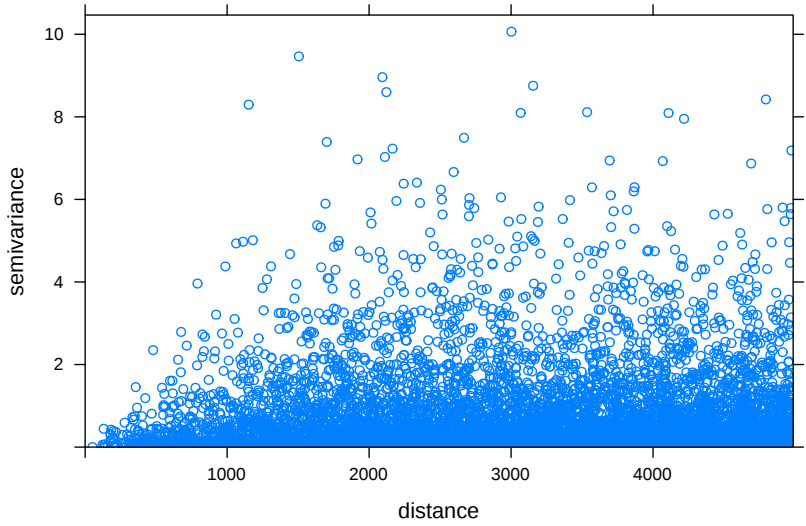
```
moja_chmura = variogram(tavg ~ 1, moje_punkty, cloud = TRUE)
plot(moja_chmura)
```

## C.3. Tworzenie modeli semiwariogramów

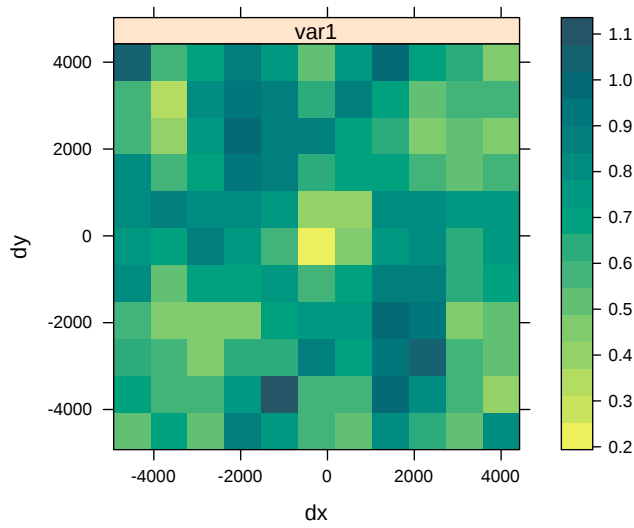
Posiadając już poprawne dane można sprawdzić czy badane zjawisko wykazuje anizotropię przestrzenną poprzez stworzenie mapy semiwariogramu (rycina C.6).

```
moja_mapa = variogram(tavg ~ 1,
                      locations = moje_punkty,
                      cutoff = 4500,
                      width = 850,
                      map = TRUE)
plot(moja_mapa, threshold = 30,
     col.regions = hcl.colors(40, palette = "ag_GrnYl", rev = TRUE))
```

Uzyskana mapa nie pozwala na jednoznaczne stwierdzenie kierunkowej zmienności podobieństwa badanej cechy, w związku z tym można skupić



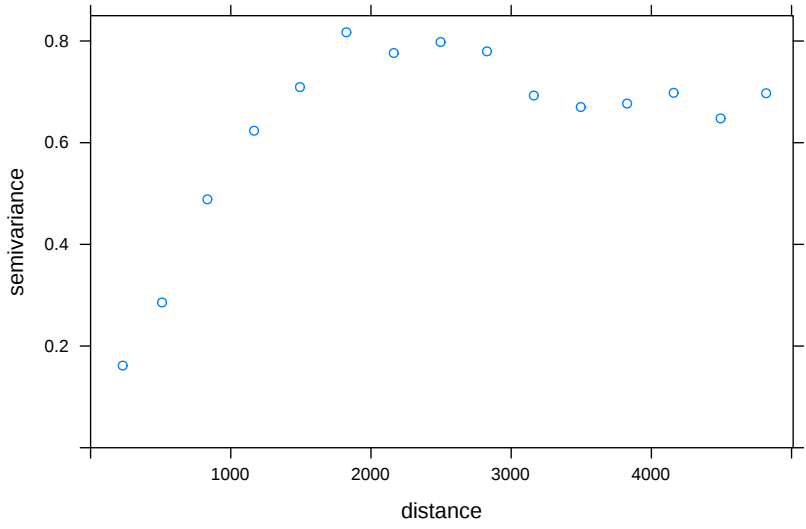

Rycina C.5.: Chmura semiwariogramu zmiennej tavg.




Rycina C.6.: Mapa semiwariogramu zmiennej tavg.

się na modelowaniu izotropowym. Kolejnym etapem jest stworzenie semiwariogramu oraz jego modelowanie. Optymalnie tworzy się więcej niż jeden 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 (ryciny C.7, C.8, C.9, C.10).

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

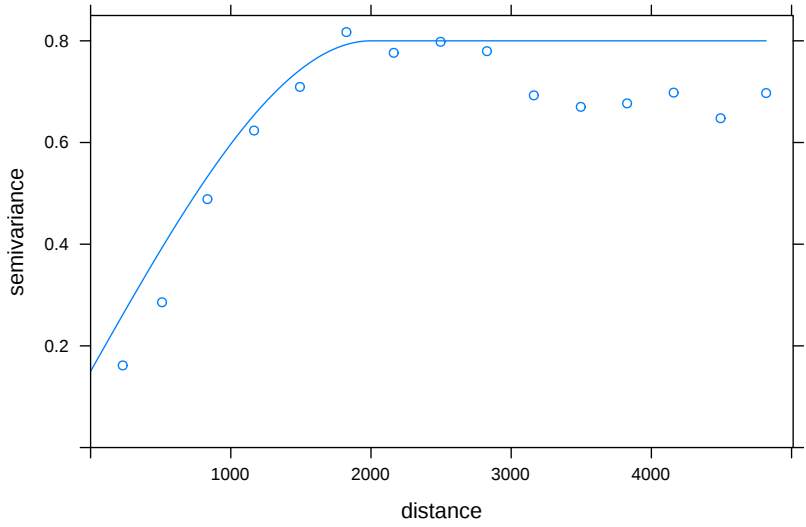



Rycina C.7.: Semiwariogram zmiennej tavg.

```
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()`.

```
moje_punkty$X = st_coordinates(moje_punkty)[, 1]
moje_punkty$Y = st_coordinates(moje_punkty)[, 2]
```




Rycina C.8.: Model semiwariogramu zmiennej tavg.

```
moj_semiwar2 = variogram(tavg ~ X + Y,
                         locations = moje_punkty)
plot(moj_semiwar2)
```

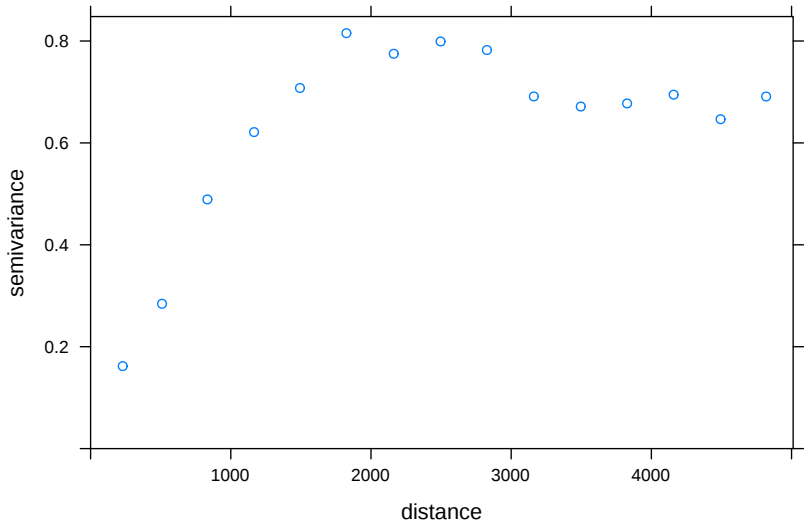
```
moj_model2 = vgm(model = "Sph", nugget = 0.1)
moj_model2 = fit.variogram(moj_semiwar2, moj_model2)
moj_model2
```

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

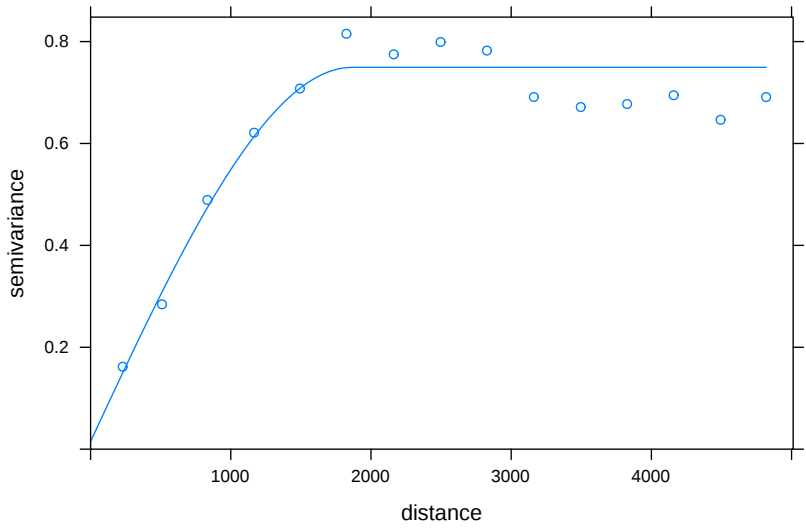
```
plot(moj_semiwar2, moj_model2)
```

## C.4. Ocena jakości modeli

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




Rycina C.9.: Semiwariogram zmiennej tavg uwzględniający współrzędne x i y.




Rycina C.10.: Model semiwariogramu zmiennej tavg uwzględniający współrzędne x i y.

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

```
ocena2 = krige.cv(tavg ~ X + Y,
                  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
```

## C.5. Stworzenie siatki

Przedostatnim krokiem jest utworzenie siatki do estymacji. Do niej zostaną wpisane wartości uzyskane z modelu semiwariogramu.

```
punkty_bbox = st_bbox(moje_punkty)
moja_siatka = st_as_stars(punkty_bbox,
                          dx = 100,
                          dy = 100)
moja_siatka$X = st_coordinates(moja_siatka)[, 1]
moja_siatka$Y = st_coordinates(moja_siatka)[, 2]
```

## C.6. Stworzenie estymacji

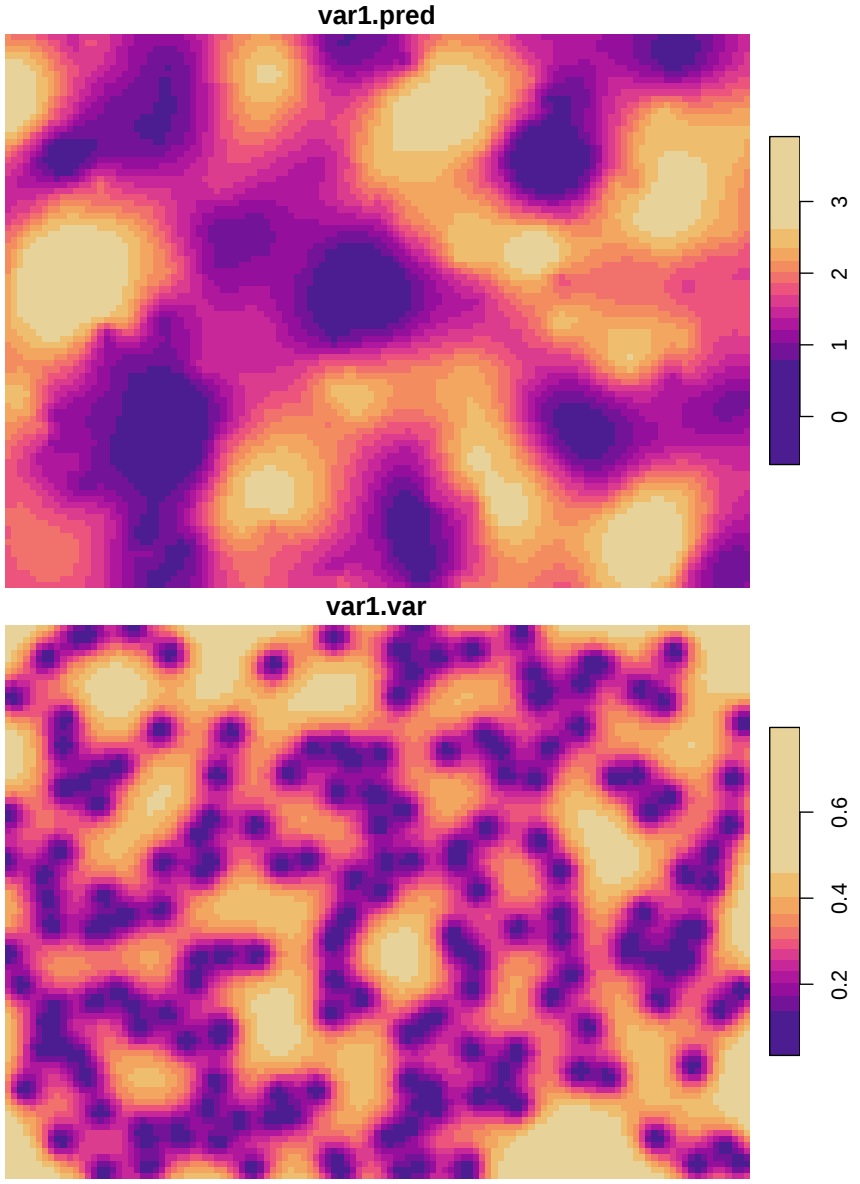
Następnie nowo utworzona siatka może posłużyć do stworzenia estymacji (rycina C.11).

```
moja_estymacja = krige(tavg ~ X + Y,
                       locations = moje_punkty,
                       newdata = moja_siatka,
                       model = moj_model2)
```

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

```
plot(moja_estymacja[1], col = paleta)
plot(moja_estymacja[2], col = paleta)
```

Wynikiem uproszczonej analizy geostatystycznej jest mapa estymowanych wartości temperatury dla całego badanego obszaru oraz mapa przedstawiająca wariancję estymacji temperatury.




Rycina C.11.: Estymacja i wariancja estymacji drugiego modelu zmiennej tavg.

# Bibliografia


Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2021). *rmarkdown: Dynamic Documents for R.* R package version 2.8.

Appelhans, T., Detsch, F., Reudenbach, C., and Woellauer, S. (2021). *mapview: Interactive Viewing of Spatial Data in R.* R package version 2.10.0.

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

Biecek, P. (2014). *Przewodnik Po Pakiecie R.* Oficyna Wydawnicza GIS.

Bivand, R. S., Pebesma, E. J., and Gomez-Rubio, V. (2014). *Applied spatial data analysis with R.* Springer.

Cressie, N. (1992). Statistics for spatial data. *Terra Nova*, 4(5):613–617.

Czernecki, B. (2018). *Metody Przetwarzania Danych Meteorologicznych w Języku Programowania R.*

Gagolewski, M. (2016). *Programowanie w Jezyku R.* Wydawnictwo Naukowe PWN.

Gillespie, C. and Lovelace, R. (2016). *Efficient R Programming: A Practical Guide to Smarter Programming.* " O'Reilly Media, Inc.".

Giraudoux, P. (2021). *pgirmess: Spatial Analysis and Data Mining for Field Ecologists.* R package version 1.7.0.

Goovaerts, P. et al. (1997). *Geostatistics for natural resources evaluation.* Oxford University Press on Demand.

Hengl, T. (2009). *A practical guide to geostatistical mapping.* Hengl Amsterdam.

Hijmans, R. J., Phillips, S., Leathwick, J., and Elith, J. (2020). *dismo: Species Distribution Modeling.* R package version 1.3-3.

Isaaks, E. H. and Srivastava, M. R. (1989). *Applied geostatistics.*

Lovelace, R., Nowosad, J., and Muenchow, J. (2019). *Geocomputation with R.* Chapman and Hall/CRC Press.

Nowosad, J. (2019). *rcartocolor: CARTOColors Palettes*. R package version 2.0.0.

Nowosad, J. (2020). *Elementarz programisty: Wstęp do programowania używając R*. Space A, Poznan.

Nowosad, J. (2021). *geostatbook: Geostatystyka w R*. R package version 3.2.0.

Nychka, D., Furrer, R., Paige, J., Sain, S., Gerber, F., and Iverson, M. (2021). *fields: Tools for Spatial Data*. R package version 12.3.

Pebesma, E. (2021a). *sf: Simple Features for R*. R package version 1.0-0.

Pebesma, E. (2021b). *stars: Spatiotemporal Arrays, Raster and Vector Data Cubes*. R package version 0.5-3.

Pebesma, E. and Graeler, B. (2021). *gstat: Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation*. R package version 2.0-7.

Pebesma, E. J. (2001). Gstat user's manual. *Dept. of Physical Geography, Utrecht University, Utrecht, The Netherlands*.

Pebesma, E. J. and Bivand, R. S. (2019). *Spatial Data Science*.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rossiter, D. (2016). An introduction to geostatistics with R/gstat.

Silge, J., Chow, F., Kuhn, M., and Wickham, H. (2021). *rsample: General Resampling Infrastructure*. R package version 0.1.0.

Stach, A. (2009). Praktyczny poradnik–jak szybko zrozumieć i wykorzystać geostatystykę w pracy badawczej.

Webster, R. and Oliver, M. A. (2007). *Geostatistics for environmental scientists*. John Wiley & Sons.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., and Dunnington, D. (2020). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.3.

Wickham, H. and Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. " O'Reilly Media, Inc.".

Xie, Y. (2021a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.22.

Xie, Y. (2021b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.33.