# Rasztermű̈veletek

## Térinformatika R-ben

2023.11.20.

Section 1

# Felbontás lekérése és megváltoztatása

# Felbontás lekérése

## Felbontás

- 2 számérték (x és y irányban)
- a vetület szerint értelmezett mértékegységben (WGS-84 esetén fok!)
- a cellaközéppontok távolsága
- vagy: a cellák élhossza

## Felbontás lekérdezése

- `xres(x)`: x irányú felbontás
- `yres(x)`: y irányú felbontás
- `res(x)`: x és y irányú felbontás

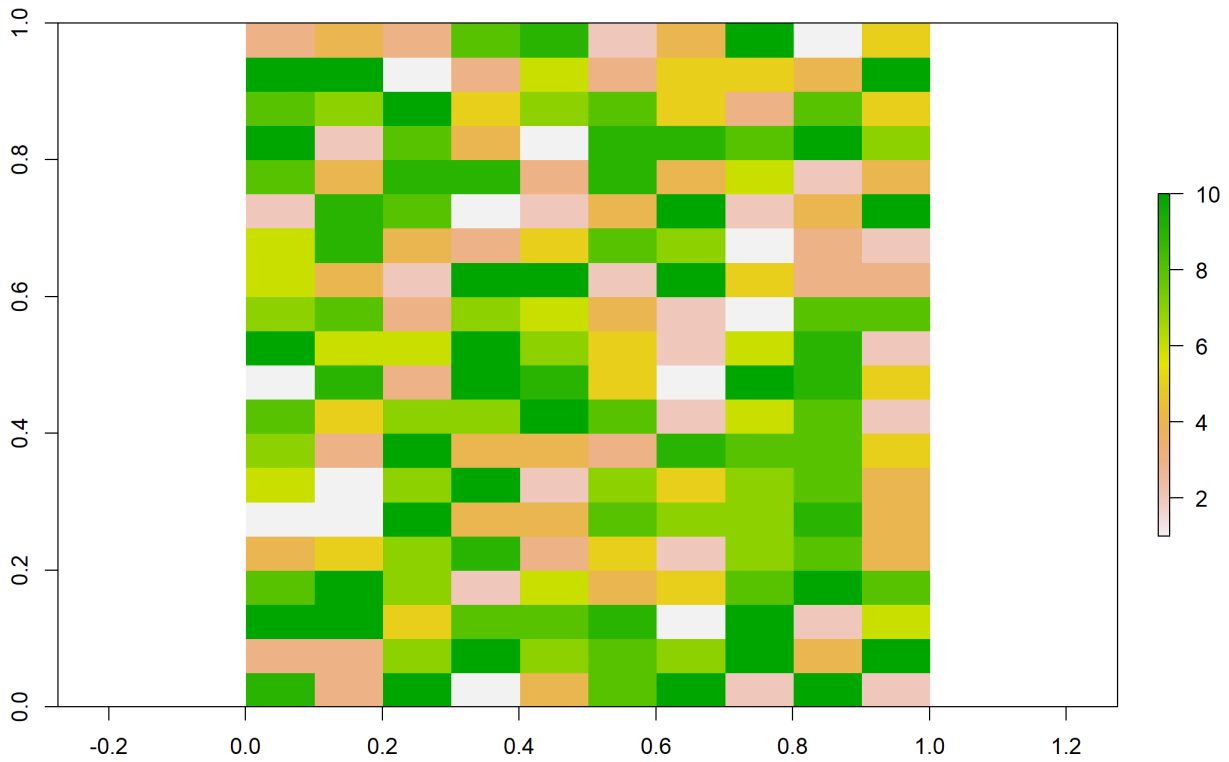
# Felbontás lekérése

```
library(raster)
```

```
set.seed(12345)
szamok <- sample(x = 1:10, size = 10 * 20, replace = TRUE)
szammatrix <- matrix(data = szamok, ncol = 10, nrow = 20)
szamraszter <- raster(x = szammatrix)
```

# Felbontás lekérése

```
plot(szamraszter)
```



# Felbontás lekérése

```
print(szamraszter)
```

```
class       : RasterLayer
dimensions  : 20, 10, 200  (nrow, ncol, ncell)
resolution  : 0.1, 0.05  (x, y)
extent      : 0, 1, 0, 1  (xmin, xmax, ymin, ymax)
crs         : NA
source      : memory
names       : layer
values      : 1, 10  (min, max)
```

# Felbontás lekérése

```
res(szamraszter)
```

```
[1] 0.10 0.05
```

```
xres(szamraszter)
```

```
[1] 0.1
```

```
yres(szamraszter)
```

```
[1] 0.05
```

# Felbontás megváltoztatása

## Megváltatoztatás módjai

- egyszerű felülírás: `res()`
- összevonás (aggregálás) és felaprózás (diszaggregálás)
- átvetítés
- interpoláció (sokféle módszer). Később...

## Felbontás egyszerű felülírása

- `res(x) <- value`
- `x`: a raszter
- `value`: egy vagy két szám
- egy szám esetén mindkét irányban ez lesz az új felbontás
- csak **üres** raszterek esetén használandó!

# Felbontás egyszerű felülírása

Üres (értékek nélküli) rasztert a `raster()` függvénnyel készíthetünk nem üres (egy- vagy többrétegű raszterből):

```
ures <- raster(szamraszter)
```

```
print(ures)
```

```
class       : RasterLayer
dimensions  : 20, 10, 200  (nrow, ncol, ncell)
resolution  : 0.1, 0.05  (x, y)
extent      : 0, 1, 0, 1  (xmin, xmax, ymin, ymax)
crs         : NA
```

Nincsenek értékek benne.

# Felbontás egyszerű felülírása

Az üres raszter felbontása, vetülete és minden egyéb tulajdonsága megegyezik az eredetivel, csak éppen nincsenek benne értékek (nincs megjeleníthető rétege).

```
res(ures)
```

```
[1] 0.10 0.05
```

```
res(ures) <- c(0.2, 0.1)
res(ures)
```

```
[1] 0.2 0.1
```

# Felbontás egyszerű felülírása

```
plot(ures)
```

```
Error in .plotraster2(x, col = col, maxpixels = maxpixels,
 add = add, : no values associated with this RasterLayer
```

Egy szám megadása esetén mindkét dimenzióban ez lesz az új felbontásunk:

```
res(ures) <- 0.5
res(ures)
```

```
[1] 0.5 0.5
```

Ha a raszterünk nem üres (tartalmaz értékeket), más módszerhez kell folyamodnunk...

# Összevonás

## Összevonás (aggregálás)

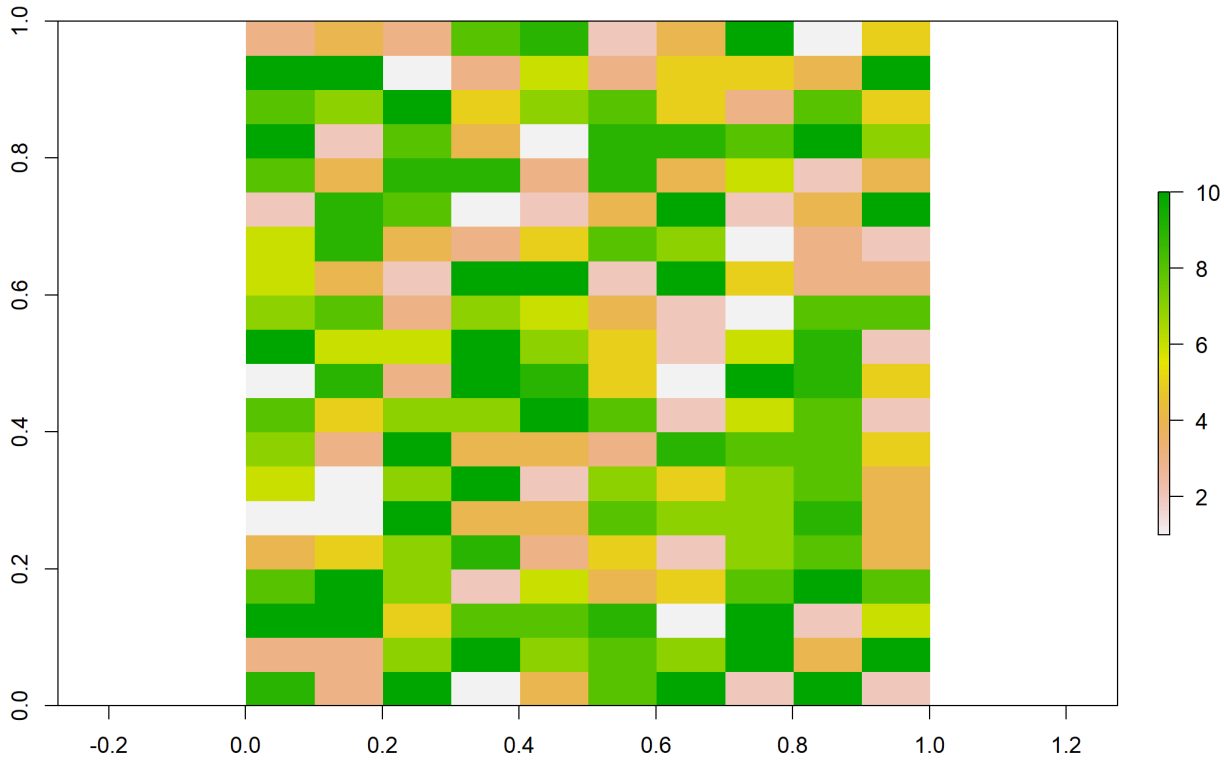
- szomszédos cellákat blokkokban összevonjuk
- a felbontás az eredeti felbontás egész számú többszöröse lesz
- az egy blokkba tartozó cellák értékei egybeolvadnak
- az egybeolvasztás függvénye tetszőleges
- de mindenképpen adatvesztés történik

## `aggregate(x, fact = 2, fun = mean)`

- `x`: a túl finom felbontású raszter
- `fact`: az összevonás mértéke (egy vagy két szám). Alapértelmezetten 2×2-es blokkokat aggregál
- `fun`: az új cellaértéket a régiekből kiszámító függvény

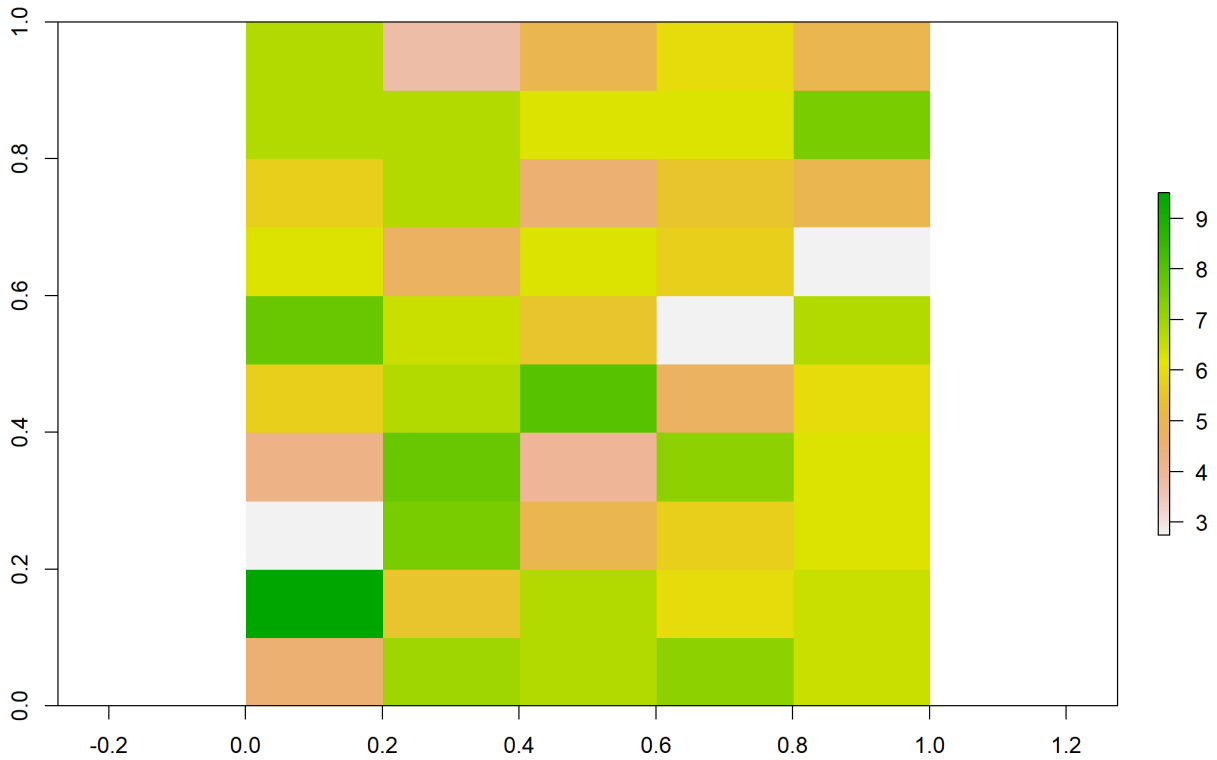
# Összevonás

```
szamraszter <- raster(x = szammatrix)
plot(szamraszter)
```



# Összevonás

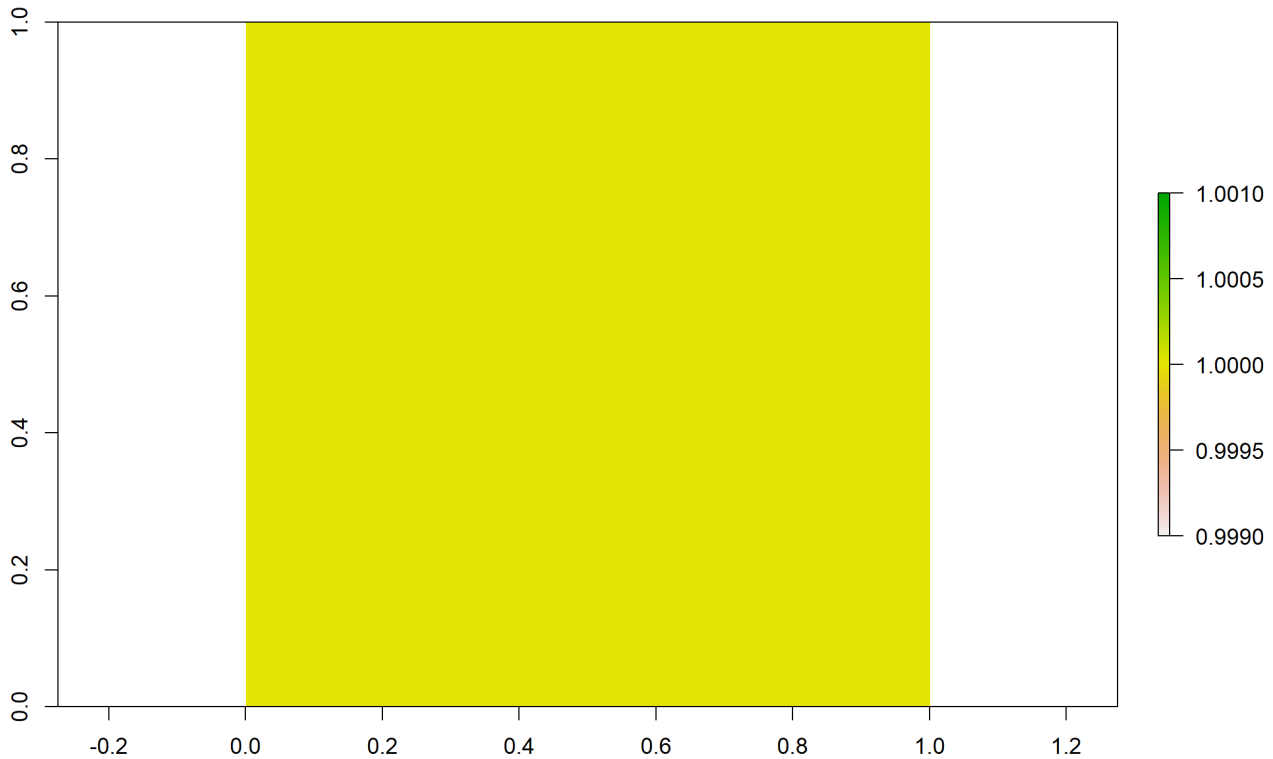
```
osszevont1 <- aggregate(x = szamraszter, fact = 2, fun =
 mean)
plot(osszevont1)
```



# Összevonás

Az aggregáló függvény bármi lehet, ami sok számból egyet képez.

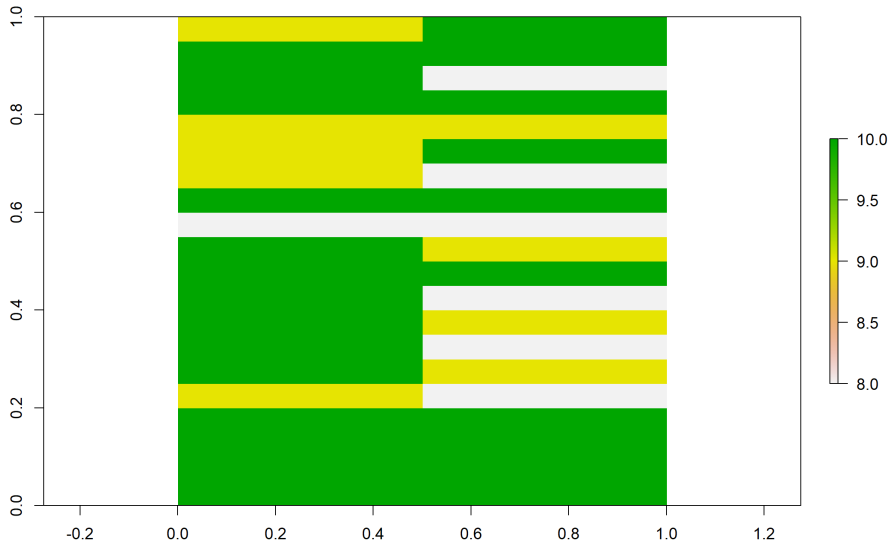
```
osszevont2 <- aggregate(x = szamraszter, fact = 5, fun =
 min)
plot(osszevont2)
```



# Összevonás

A ket dimenzió mentén eltérő összevonási tényezőket is használhatunk. Akár az érték lehet 1 is (értsd: abban a dimenzióban nem történik összevonás).

```
osszevont3 <- aggregate(x = szamraszter, fact = c(5, 1),
 fun = max)
plot(osszevont3)
```



# Felaprózás

## Felaprózás (diszaggregálás)

- finomabb felbontású rasztert hozunk létre
- a felbontást mindkét irányban egész számmal osztjuk
- adat nem vész el
- de az új cellák értéke korábban nem volt ismert (ki kell találni)

## `disaggregate(x, fact, method)`

- `x`: a túl durva felbontású raszter
- `fact`: a felaprózás mértéke (egy vagy két szám)
- `method`: opcionális. Ha üresen hagyjuk, az eredeti cellaértéket másolja.
- `method = "bilinear"`: bilineáris interpolációt végez (elken/átlagol)

# Felaprózás

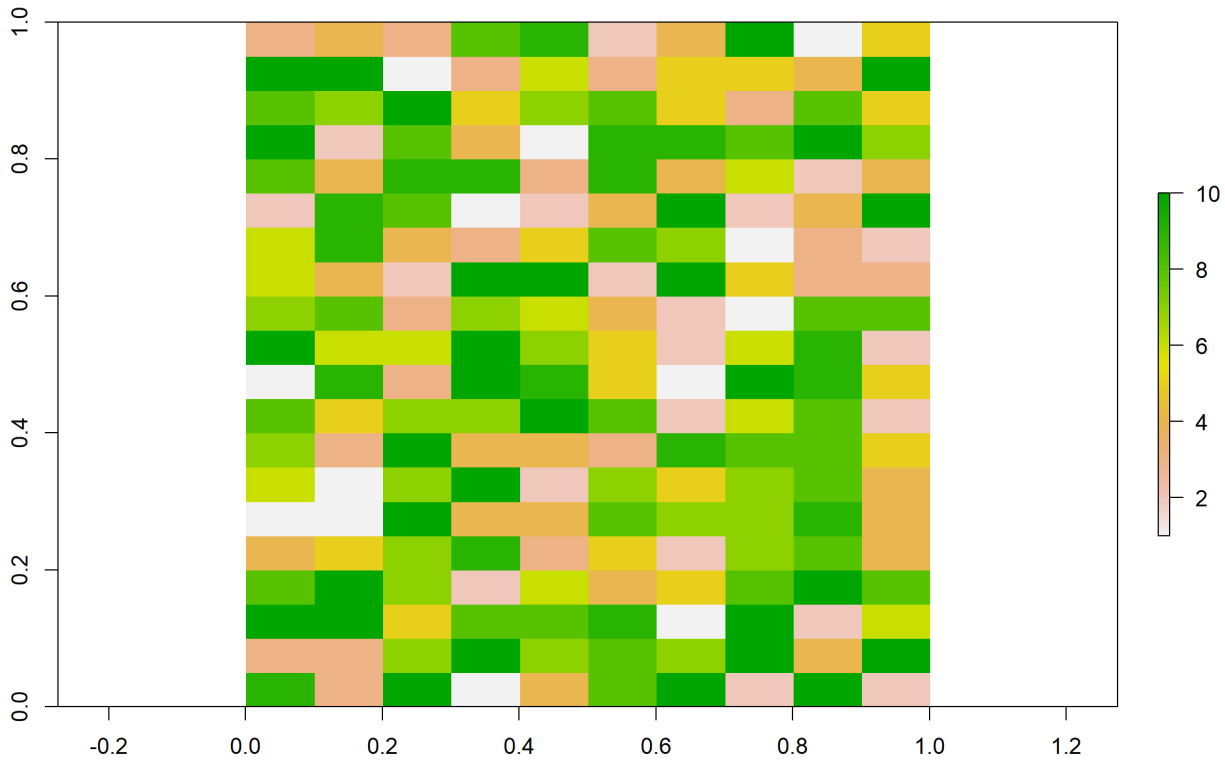
```
ncell(szamraszter)
```

```
[1] 200
```

```
felaprozott1 <- disaggregate(x = szamraszter, fact = 2)
```

# Felaprózás

```
plot(felaprozott1)
```




Sok különbséget nem látunk...

# Felaprózás

De a cellaszám megmutatja, hogy a felbontás finomodott:

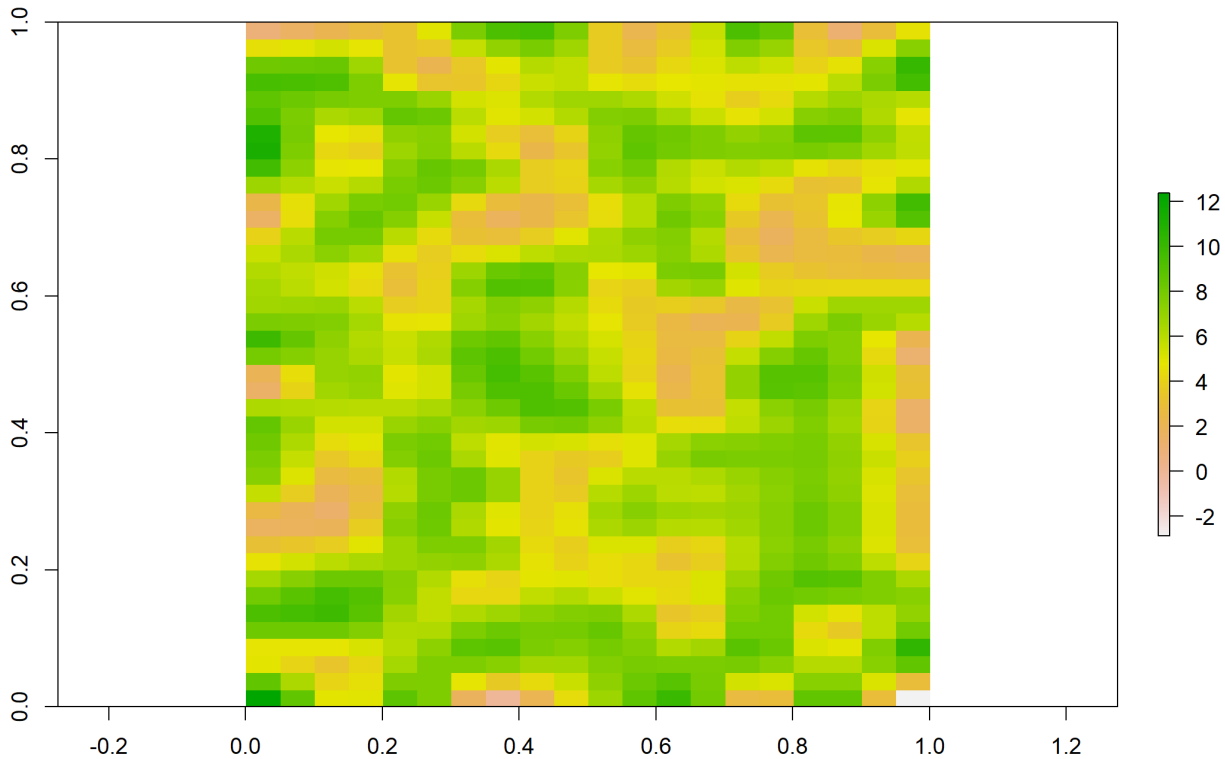
```
ncell(felaprozott1)
```

```
[1] 800
```

# Felaprózás

Bilineáris interpoláció:

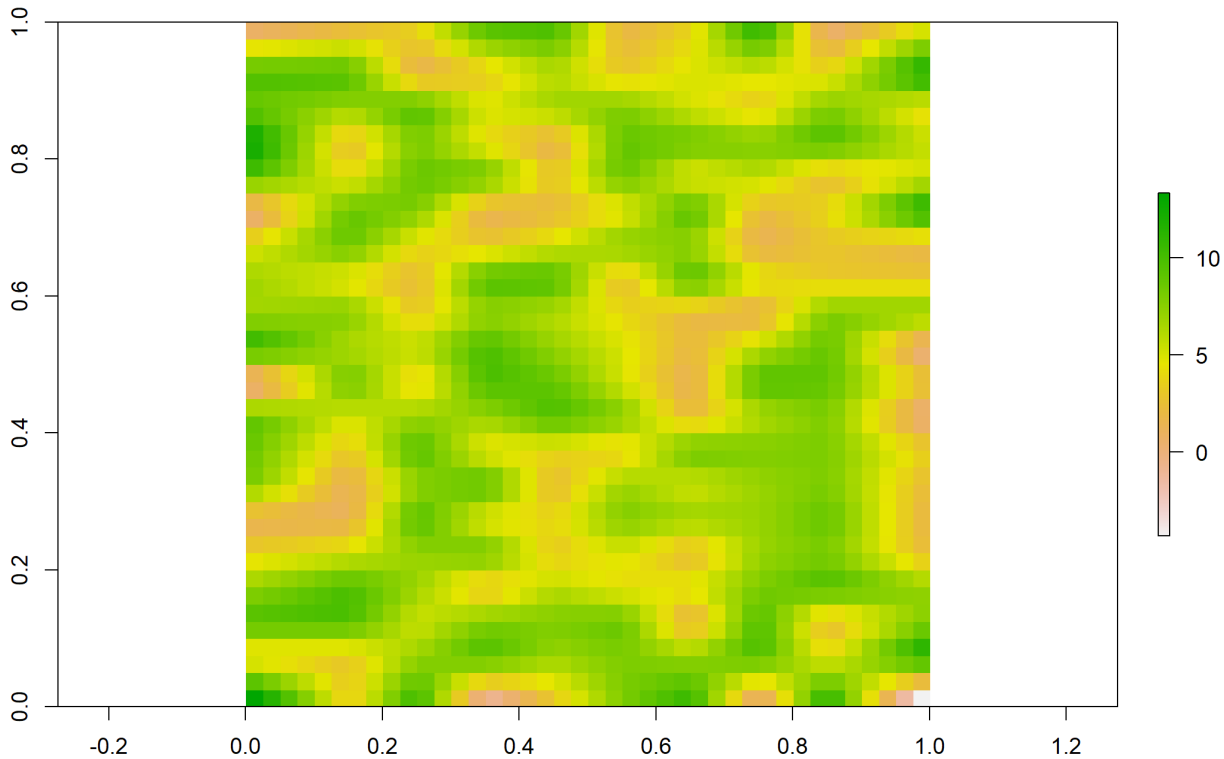
```
felaprozott2 <- disaggregate(x = szamraszter, fact = 2,
 method = "bilinear")
plot(felaprozott2)
```



# Felaprózás

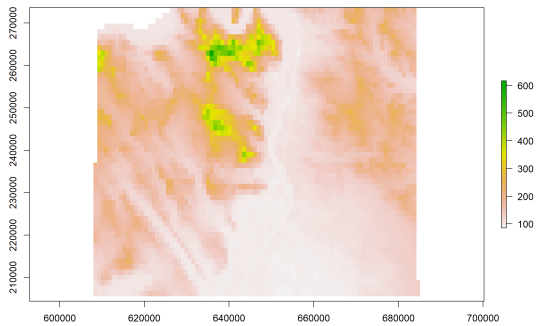
A két dimenzió mentén eltérő mértékű finomítás is alkalmazható.

```
felaprozott3 <- disaggregate(x = szamraszter, fact = c(4,
 2), method = "bilinear")
plot(felaprozott3)
```



# 1. feladat (órai)

- Olvasd be a “domborzatmodell.tif” fájlt “dem” néven `RasterLayer`ként.
- Kérd le az `x`, majd az `y` irányú felbontását.
- Hozz létre “dem2” néven egy üres rasztert a segítségével.
- Növeld meg e másolat felbontását egyszerű felülírással a 10-szeresére. (A 10-szeres értéket ne kézzel írd be...)
- Ellenőrzésképpen jelenítsd meg a legfontosabb adatait.
- Aggregáld a “dem” celláit minimumképzéssel, $10 \times 10$-es blokkokban, “dem3” néven.
- Megegyezik a dem2 és dem3 felbontása? (Ne szemmel vesd össze...)
- Jelenítsd meg az aggregált rasztert.



# 1. feladat (órai) – megoldás

```
dem <- raster(x = "domborzatmodell.tif")
```

```
xres(dem)
```

```
[1] 85.7
```

```
yres(dem)
```

```
[1] 126
```

```
dem2 <- raster(dem)
res(dem2) <- res(dem2) * 10
```

# 1. feladat (órai) – megoldás

```
print(dem2)
```

```
class       : RasterLayer
dimensions : 54, 90, 4860  (nrow, ncol, ncell)
resolution : 857, 1260  (x, y)
extent : 607957.6, 685087.6, 205637.8, 273677.8 (xmin,
 xmax, ymin, ymax)
crs : +proj=somerc +lat_0=47.1443937222222
 +lon_0=19.0485717777778 +k_0=0.99993 +x_0=650000
 +y_0=200000 +ellps=GRS67 +units=m +no_defs
```

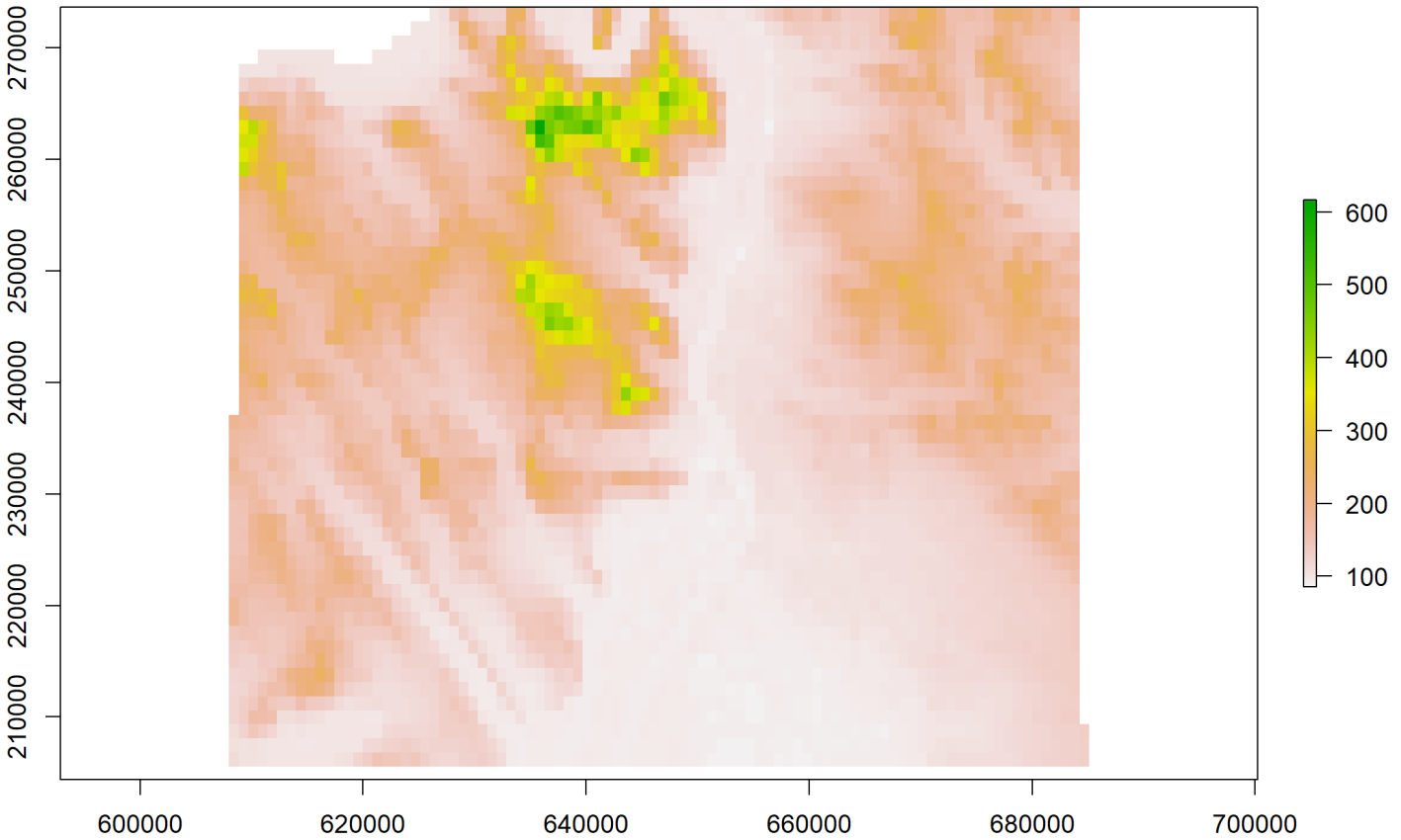
```
dem3 <- aggregate(x = dem, fact = 10, fun = min)
```

```
all(res(dem2) == res(dem3))
```

```
[1] TRUE
```

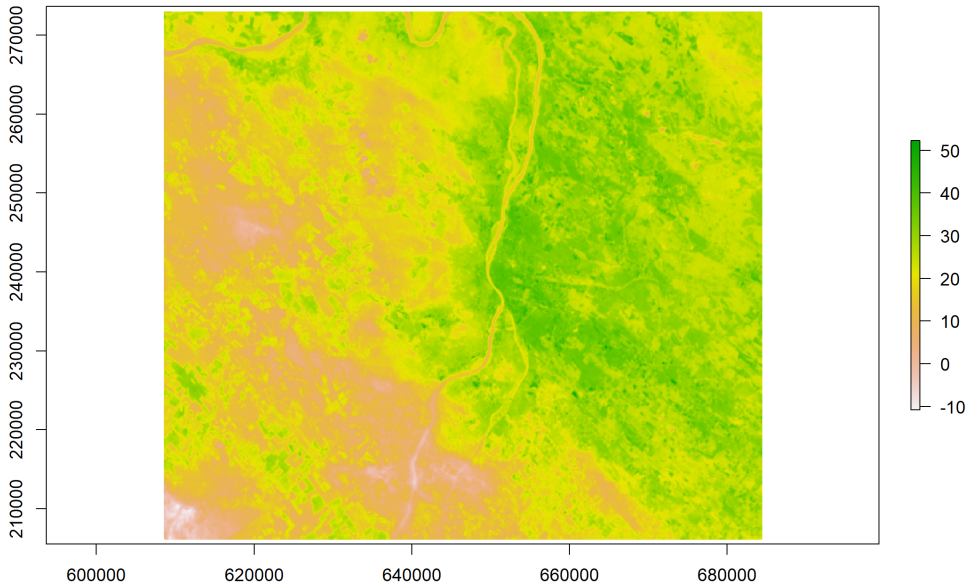
# 1. feladat (órai) – megoldás

```
plot(dem3)
```



# 2. feladat (házi)

- Olvasd be a “felszinhomerseklet.RData” fájlt.
- Hány cellája van?
- Finomítsd a felbontását mindkét irányban 2-szeres szorzóval, bilineáris interpolációt alkalmazva.
- Hány cellája van a felaprózott raszternek?
- Ellenőrzésképpen jelenítsd meg.



# 2. feladat (házi) – megoldás

```
felszinhomerseklet <- raster("felszinhomerseklet.tif")
```

```
ncell(felszinhomerseklet)
```

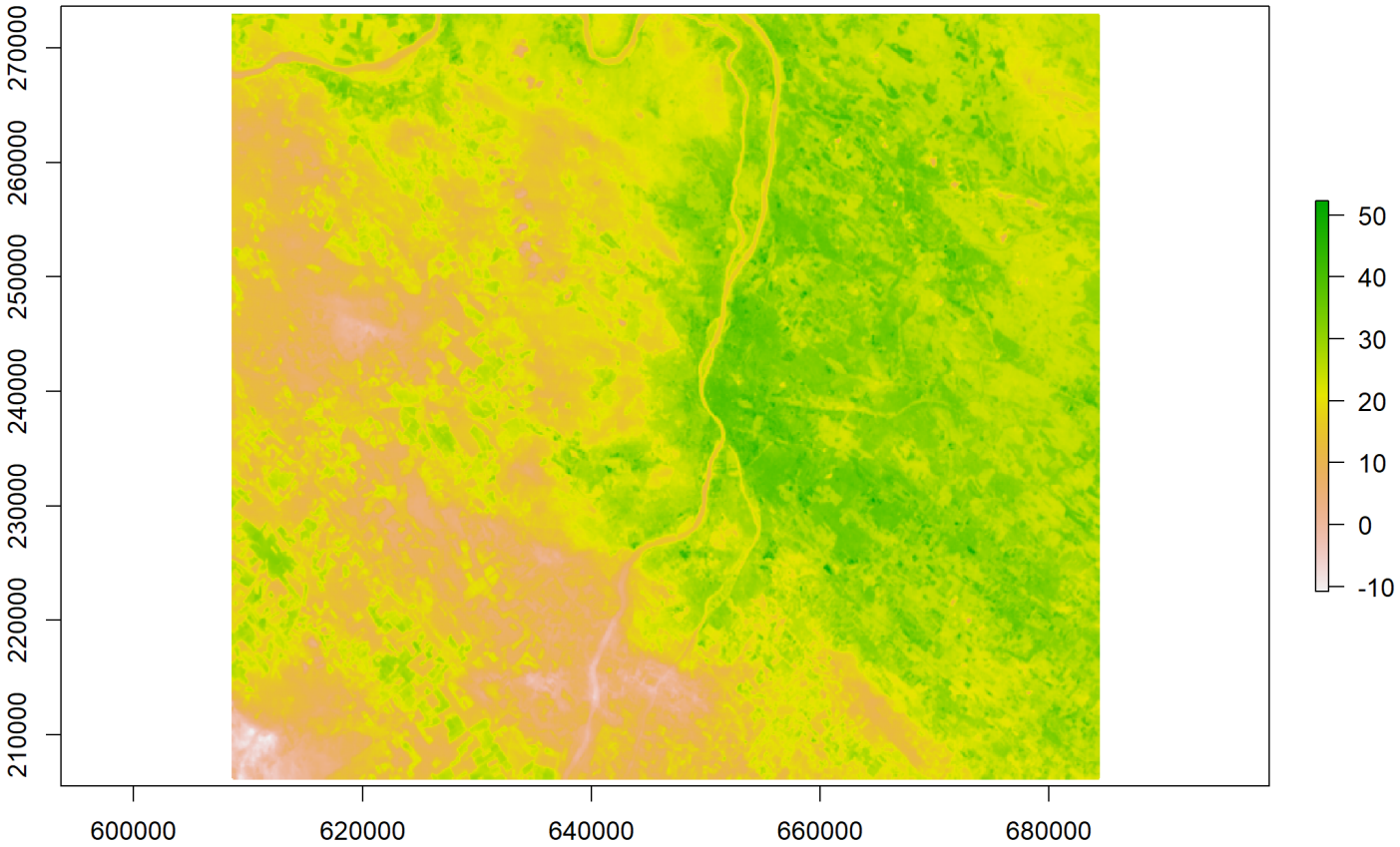
```
[1] 485818
```

```
felszinhomerseklet2 <- disaggregate(x =
 felszinhomerseklet, fact = 2, method = "bilinear")
ncell(felszinhomerseklet2)
```

```
[1] 1943272
```

# 2. feladat (házi) – megoldás

```
plot(felszinhomerseklet2)
```



Section 2

# Vetület és átvetítés

# Vetület lekérdezése és módosítása

## Raszterek vetülete

- hasonló a vektorok vetületéhez
- de kicsit fapadosabb
- nem lehet csak az EPSG-számot használni
- hanem a PROJ.4-szöveget kell ismerni

## Vetület lekérdezése és módosítása

- `projection(x)`
- `projection(x) <- value`

## Módosítás lehetséges okai

- ismeretlen vetület megadása (pl. nem olvasta be a vetületet a fájlból)
- rossz vetület javítása
- betűre pontos egyezés garantálása (egyes függvények megkövetelik)
- egyéb esetekben átvetítést alkalmazzunk!

# Vetület lekérdezése és módosítása

```
projection(dem)
```

```
[1] "+proj=somerc +lat_0=47.1443937222222
 +lon_0=19.0485717777778 +k_0=0.99993 +x_0=650000
 +y_0=200000 +ellps=GRS67 +units=m +no_defs"
```

Rögzítsük, hogy később fel tudjuk használni!

```
vetulet_eov <- projection(dem)
```

# Vetület lekérdezése és módosítása

```
projection(szamraszter)
```

```
[1] NA
```

```
projection(szamraszter) <- vetulet_eov
```

```
projection(szamraszter)
```

```
[1] "+proj=somerc +lat_0=47.1443937222222
 +lon_0=19.0485717777778 +k_0=0.99993 +x_0=650000
 +y_0=200000 +ellps=GRS67 +units=m +no_defs"
```

A módosítás sikeres. Más kérdés, hogy volt-e bármi értelme...

# Átvetítés

A szabályos rácsstruktúra garantált az átvetített raszterben is

- $\rightarrow$ ezért a cellaközéppontok elmozdulnak
- $\rightarrow$ ha a cella elmozdul, feltehetően az értéke is változik
- $\rightarrow$ az új értéket viszont nem ismerjük
- $\rightarrow$ ki kell számolni

Megadható a `method` paraméter, amellyel az új cellaértékek kiszámításának módját határozzuk meg:

- `method = "bilinear"`: bilineáris interpolációt végez. Alapértelmezett, de nagyon lassú.
- `method = "ngb"`: legközelebbi szomszéd (nearest neighbor) módszere. Az új cellához legközelebb eső korábbi cella értékét másolja.

Az átvetítést általában kétféle célra használjuk.

# Átvetítés

## Átvetítés a vetület megváltoztatása céljából

```
projectRaster(from, res, crs, method = "bilinear")
```

- `from`: átvetítendő raszter
- `res`: opcionális. A kívánt felbontás (egy vagy két szám)
- `crs`: a kívánt vetület PROJ.4-szövegként

## Átvetítés a rácsháló megváltoztatása céljából

```
projectRaster(from, to, method = "bilinear")
```

- `from`: átvetítendő raszter
- `to`: egy másik (akár üres) raszter, amelynek a rácsházóját (kiterjedés, vetület, felbontás) használni szeretnénk
- így két rasztert egymáshoz tudunk igazítani

# Átvetítés

Olvassuk be a globális csapadékrasztert, hogy kivehessük belőle a WGS-84 vetület PROJ.4-szövegét!

```
csapadek <- raster("csapadek.tif")
```

```
projection(csapadek)
```

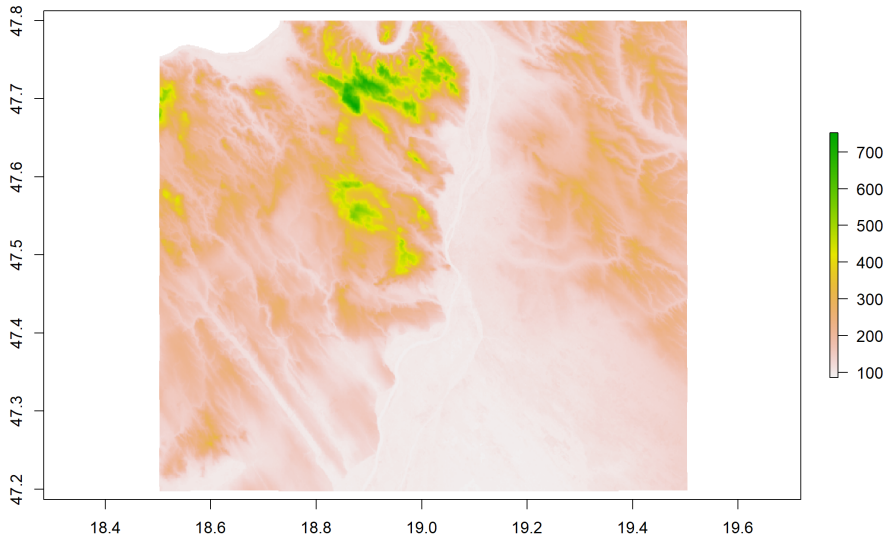
```
[1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
vetulet_wgs <- projection(csapadek)
```

# Átvetítés

```
dem_wgs1 <- projectRaster(from = dem, crs = vetulet_wgs,
 method = "ngb")
```

```
plot(dem_wgs1)
```




```
res(dem_wgs1)
```

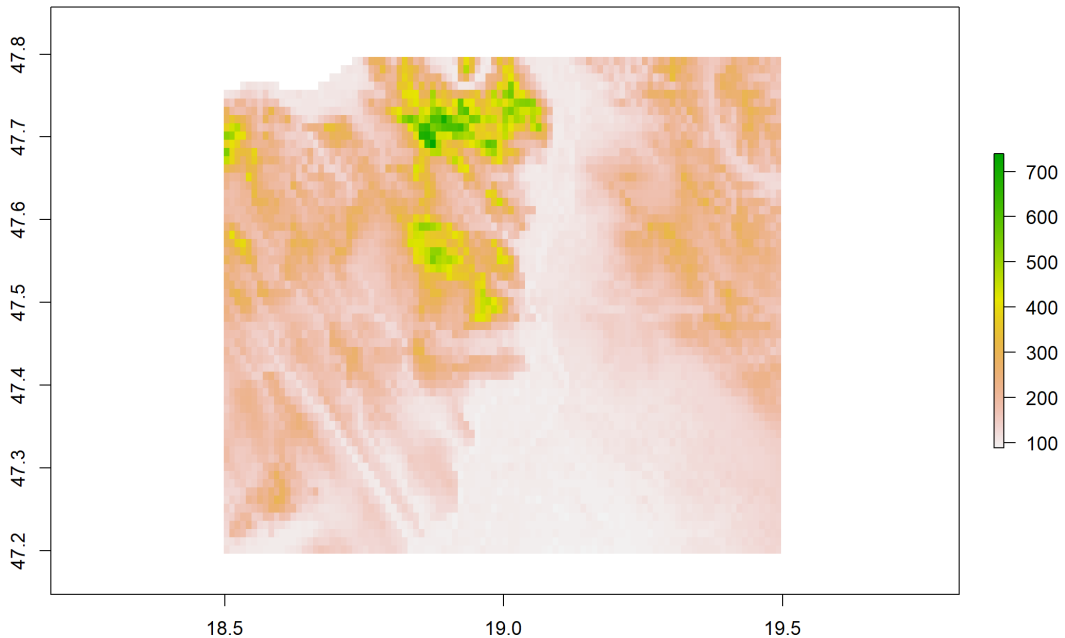
```
[1] 0.00114 0.00113
```

# Átvetítés

Rögzített felbontással:

```
dem_wgs2 <- projectRaster(from = dem, crs = vetulet_wgs,
 res = 0.01, method = "ngb")
```

```
plot(dem_wgs2)
```



# Átvetítés

```
res(dem_wgs2)
```

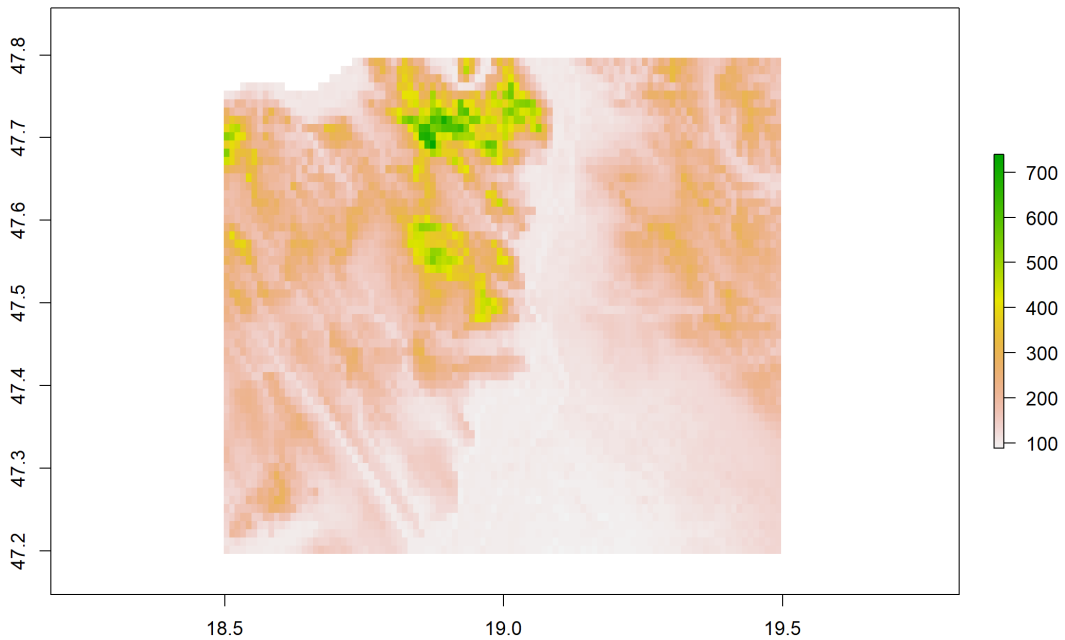
```
[1] 0.01 0.01
```

# Átvetítés

Bilineáris interpoláció:

```
dem_wgs3 <- projectRaster(from = dem, crs = vetulet_wgs,
 res = 0.01, method = "bilinear")
```

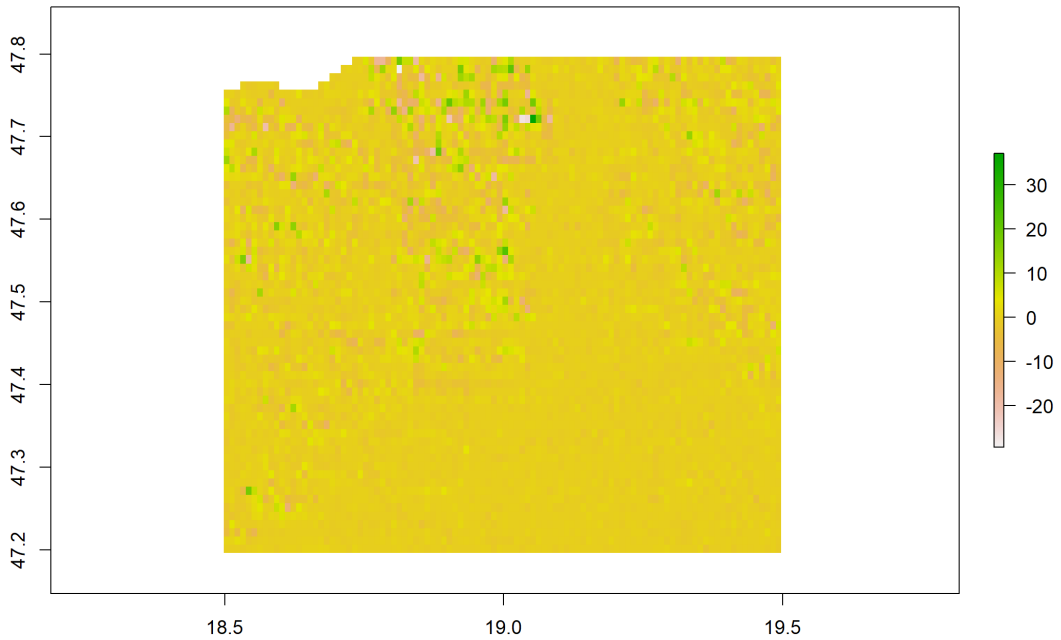
```
plot(dem_wgs3)
```



# Átvetítés

Van-e különbség a két módszer között (a számítási időn túl)?

```
elteres <- dem_wgs2 - dem_wgs3
plot(elteres)
```

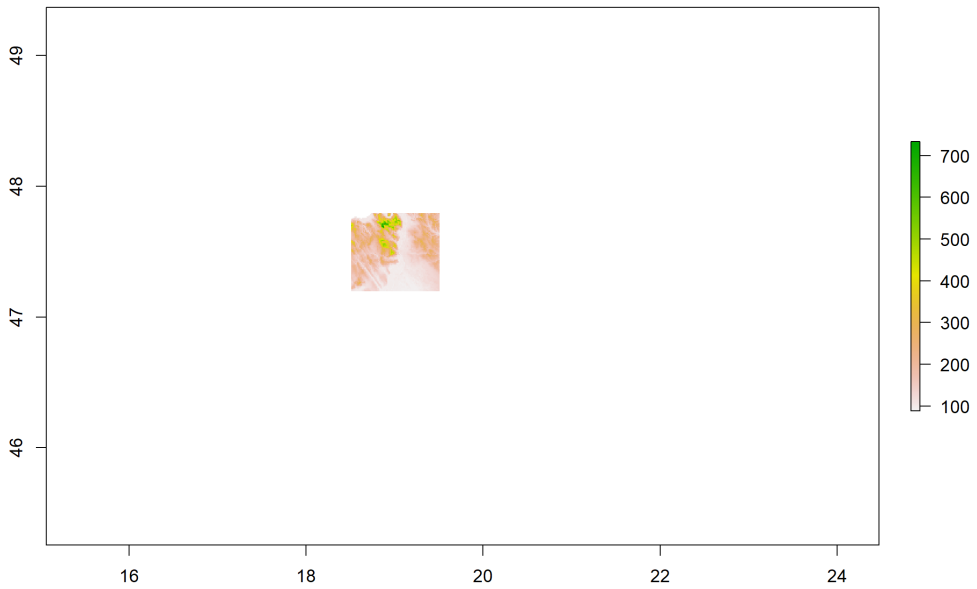


# Átvetítés

Domborzatmodell illesztése a felszínborítási raszterhez:

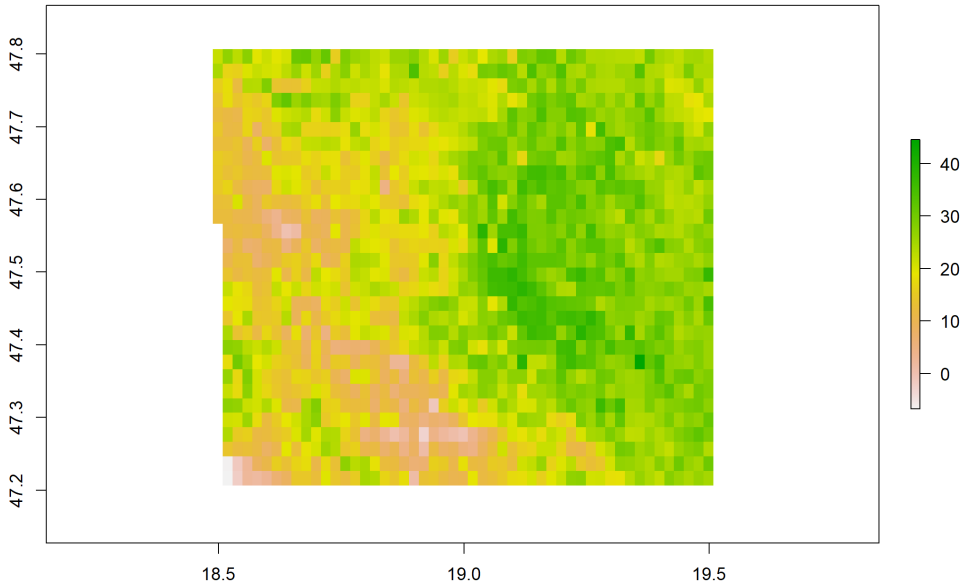
```
felszinboritas <- raster("felszinboritas.tif")
```

```
dem_igazitott <- projectRaster(from = dem, to =
 felszinboritas, method = "ngb")
plot(dem_igazitott)
```



# 3. feladat (órai)

- Mi a “felszinhomerseklet” nevű raszter vetülete?
- A korábban létrehozott “vetulet_wgs” nevű vetületbe vetítsd át a rasztert “felszinhomerseklet_wgs” néven,
  - bilineáris interpolációt alkalmazva úgy, hogy
  - mindkét dimenzió mentén 0,02 fok legyen a felbontás.
- Jelenítsd meg az átvetített rasztert.
- Mi a felbontása?



# 3. feladat (órai) – megoldás

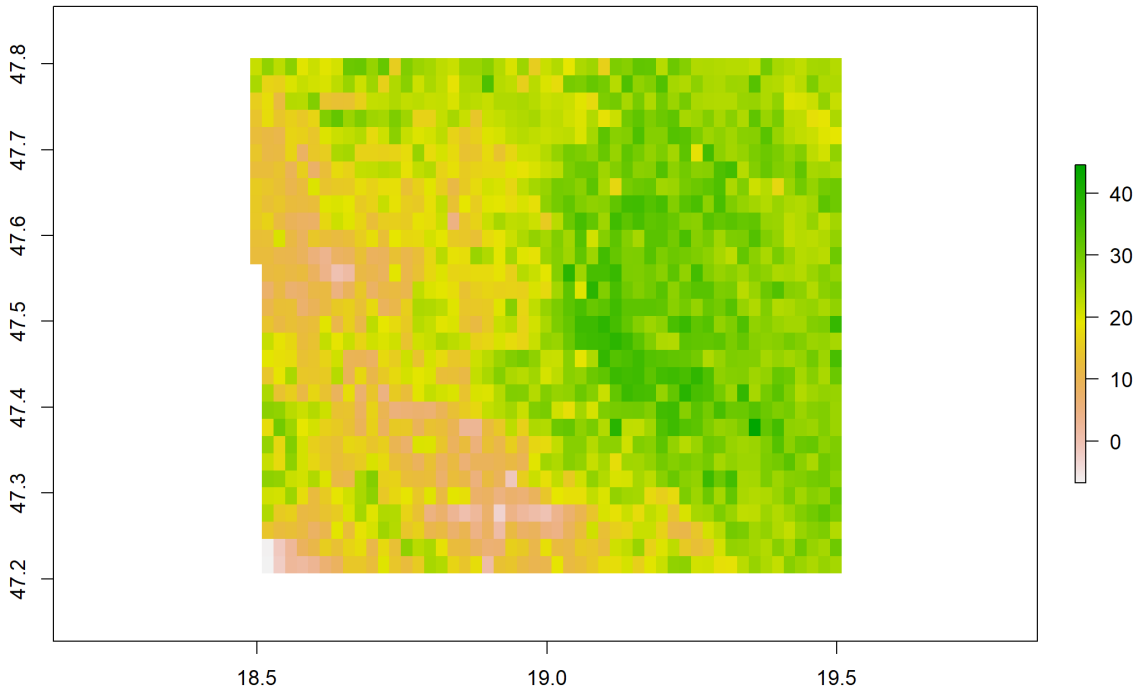
```
projection(felszinhomerseklet)
```

```
[1] "+proj=somerc +lat_0=47.1443937222222
 +lon_0=19.0485717777778 +k_0=0.99993 +x_0=650000
 +y_0=200000 +ellps=GRS67 +units=m +no_defs"
```

```
felszinhomerseklet_wgs <- projectRaster(from =
 felszinhomerseklet, crs = vetulet_wgs, res = 0.02, method
 = "bilinear")
```

# 3. feladat (órai) – megoldás

```
plot(felszinhomerseklet_wgs)
```

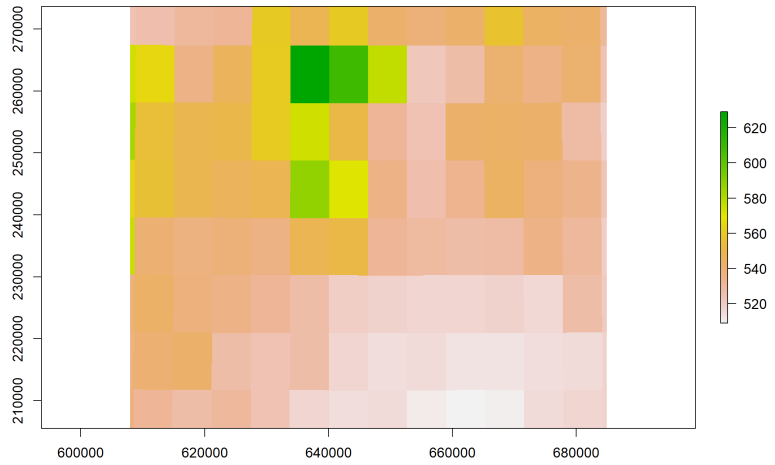


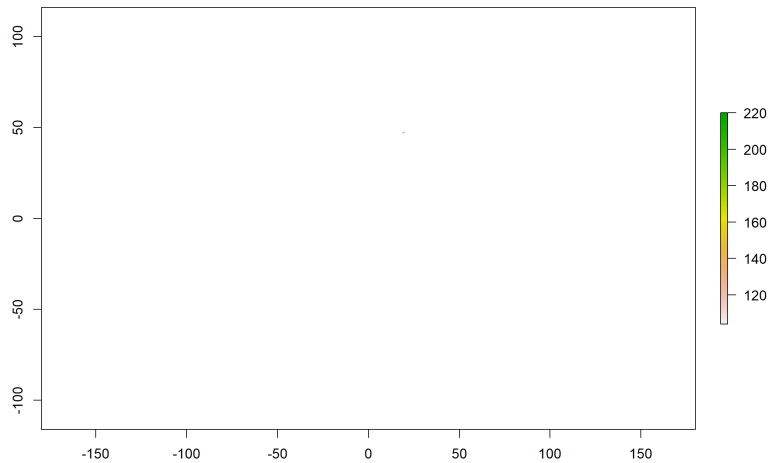

```
res(felszinhomerseklet_wgs)
```

```
[1] 0.02 0.02
```

# 4. feladat (házi)

- Hozz létre egy új rasztert a “csapadek” nevű `RasterLayer`ből úgy, hogy a legközelebbi korábbi cella értékét másolva átvetíted a pontjait a “dem” nevű raszter rácshálójába.
- Jelenítsd meg az eredményt.
- Végezd el ugyanezt fordítva, tehát a domborzatmodellt vetísd a csapadék rácshálójába.
- Ezt is jelenítsd meg.
- Miért nem látod az eredményt?

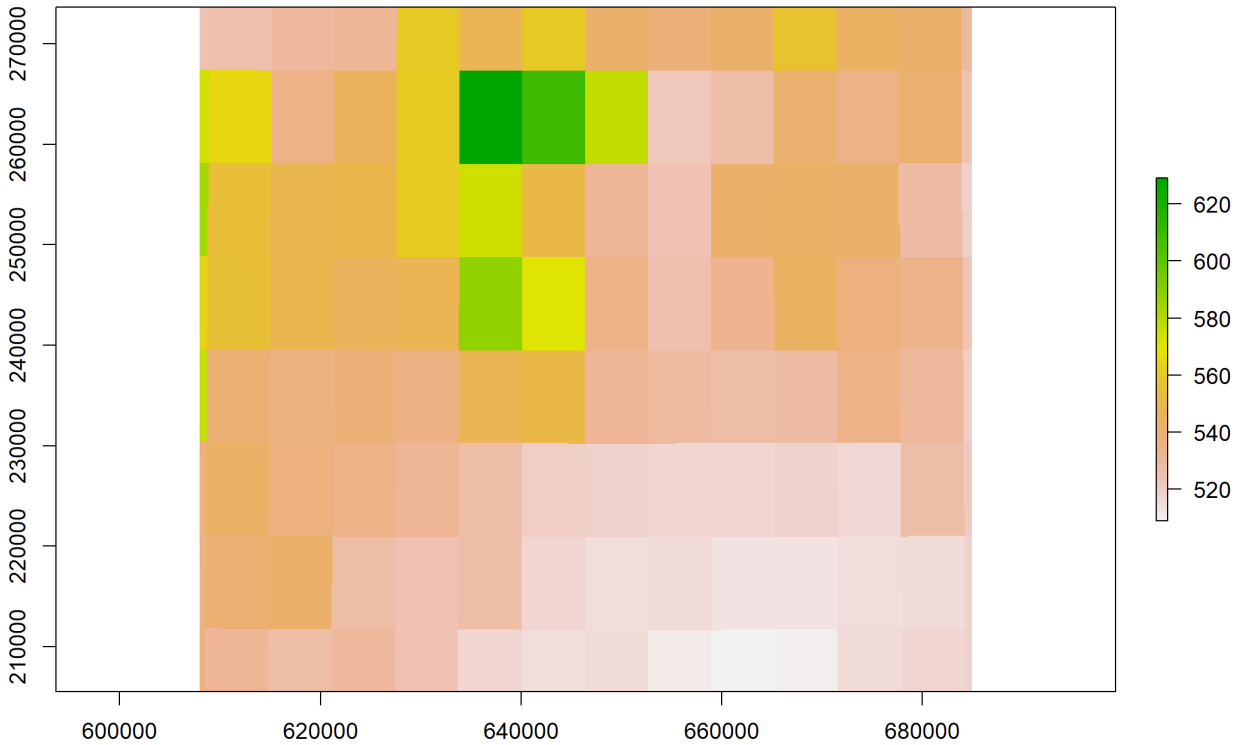


# 4. feladat (házi) – megoldás

```
csapadek_atvetitett <- projectRaster(from = csapadek, to =
 dem, method = "ngb")
```

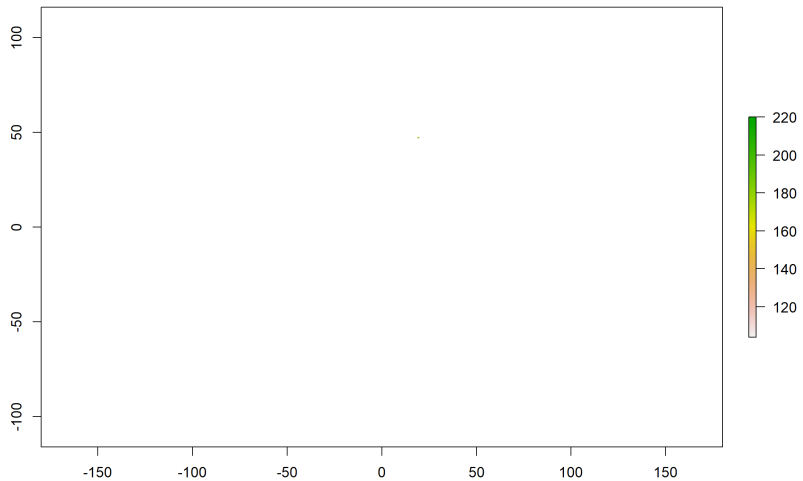
```
plot(csapadek_atvetitett)
```



# 4. feladat (házi) – megoldás

```
dem_atvetitett <- projectRaster(from = dem, to = csapadek,
 method = "ngb")
```

```
plot(dem_atvetitett)
```




Az eredmény jó, csak az egész Földet lefedő rácsháló legtöbb cellája `NA`, az ismert magasságú cellák száma olyan elenyésző, hogy ebben a felbontásban nem látszik.

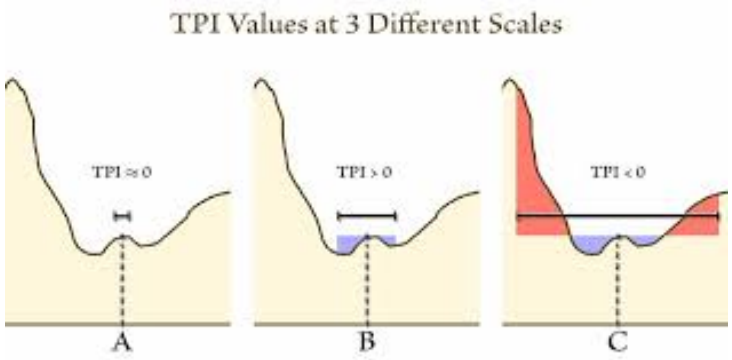
Section 3

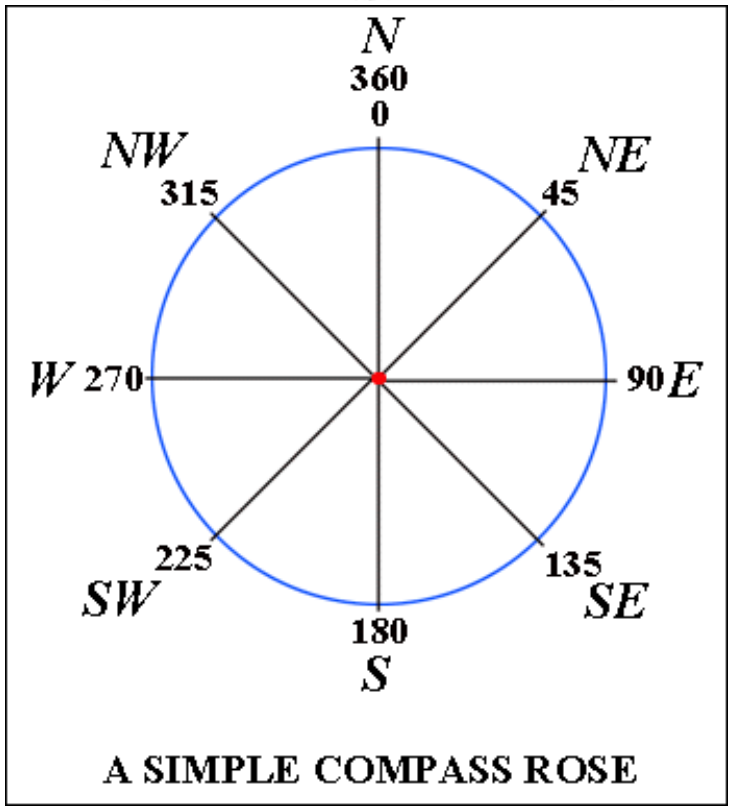
# Terepjellemzők és domborzatárnyékolás

# Terepjellemzők

## Kiszámítható terepjellemzők

- slope: lejtő meredeksége (0°: vízszintes). Ebből később lejtőkategóriát képezhetünk
- aspect: kitettség. Vagyis a lejőfelületre milyen irányból esik merőlegesen a nap?
- tri: Terrain Ruggedness Index, a felszín változatosságának mértéke
- tpi: Topographic Position Index, a környező cellákhoz képesti kiemelkedés mértéke
- és még néhány...



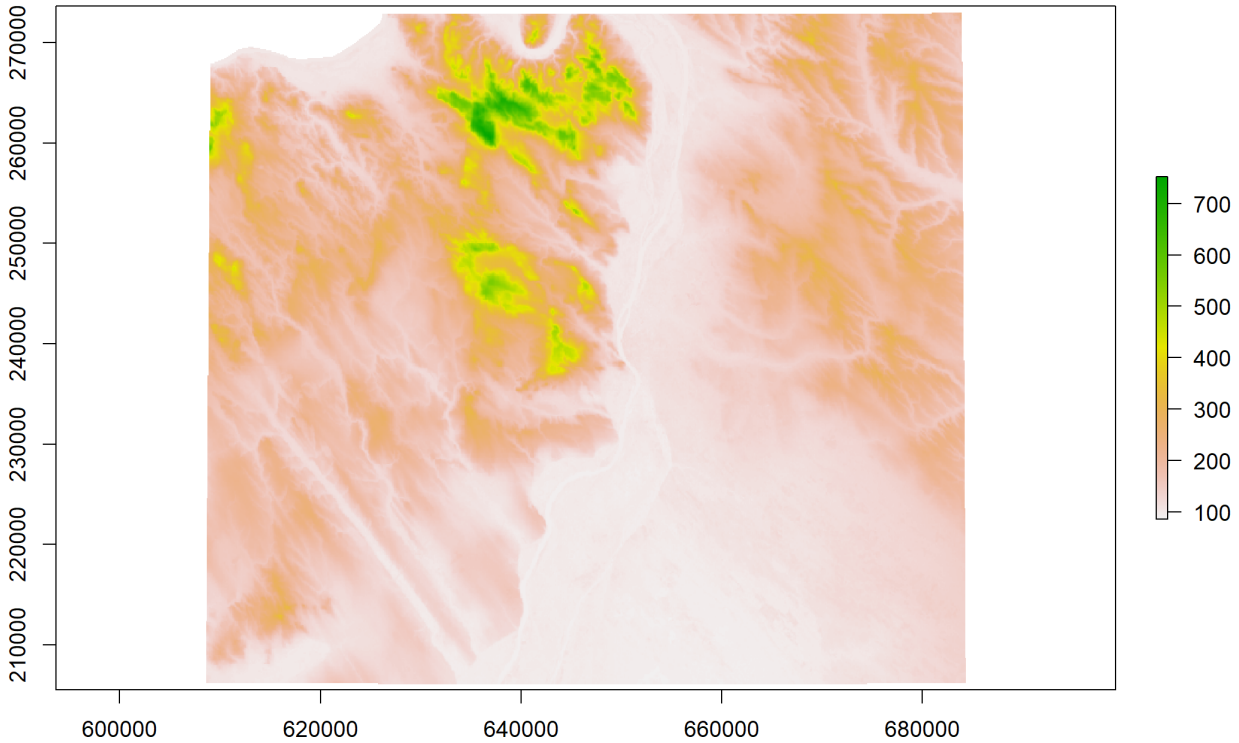
# Terepjellemzők

```
terrain(x, opt = "slope", unit = "radians")
```

- `x`: egyrétegű raszter, ami folytonos skálán értelmezhető számokat (jellemzően magasságot) tartalmaz
- `opt`: a kiszámítandó jellemző neve
- `unit`: slope/aspect esetén a szög mértékegysége: radians/degrees

# Terepjellemzők

```
plot(dem)
```



# Terepjellemzők

```
lejtoszog_radian <- terrain(x = dem, opt = "slope")
```

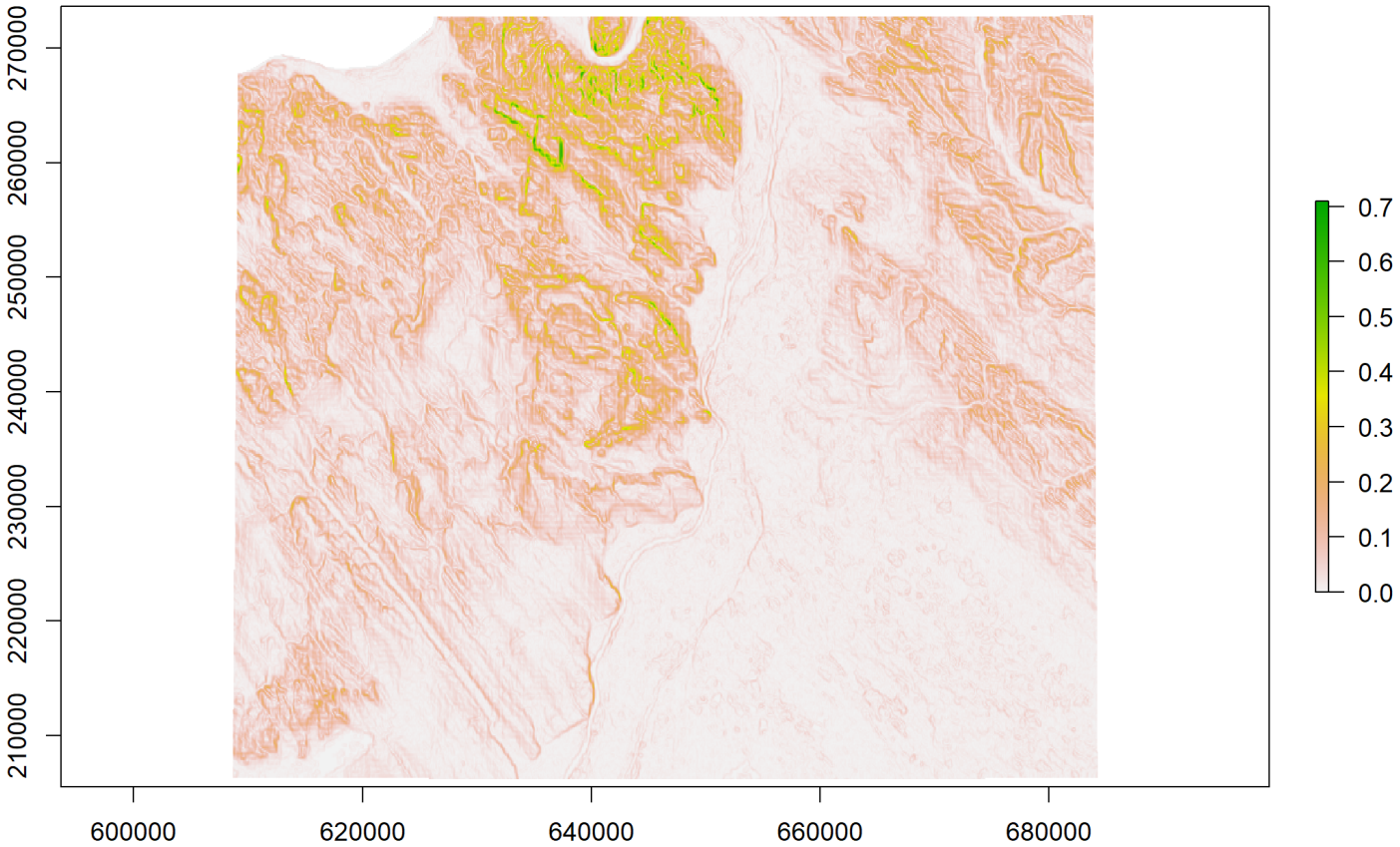
A summary() függvénnyel egy számvektor legfőbb leíró statisztikai jellemzőit kérhetjük le.

```
summary(lejtoszog_radian[])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.012   0.028   0.052   0.069   0.710   29040
```

# Terepjellemzők

```
plot(lejtoszog_radian)
```



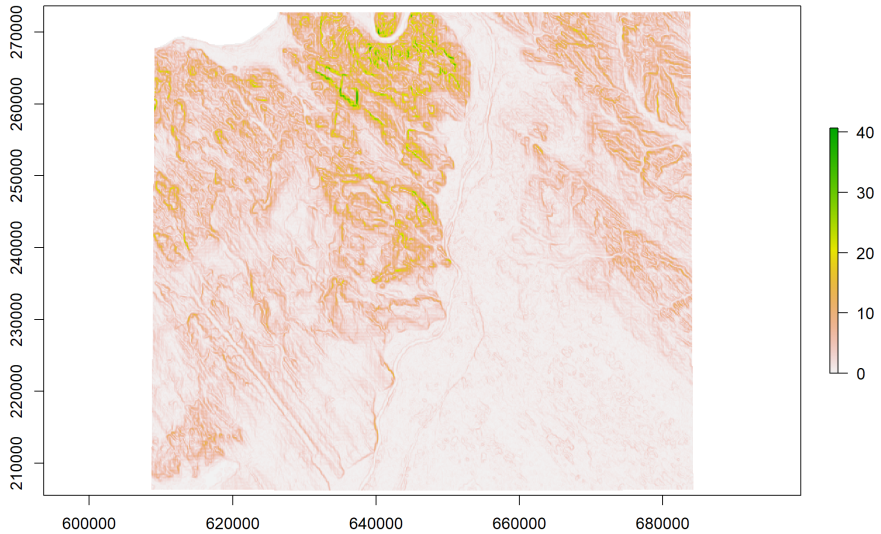
# Terepjellemzők

```
lejtoszog_fok <- terrain(x = dem, opt = "slope", unit =
 "degrees")
```

```
summary(lejtoszog_fok[])
```

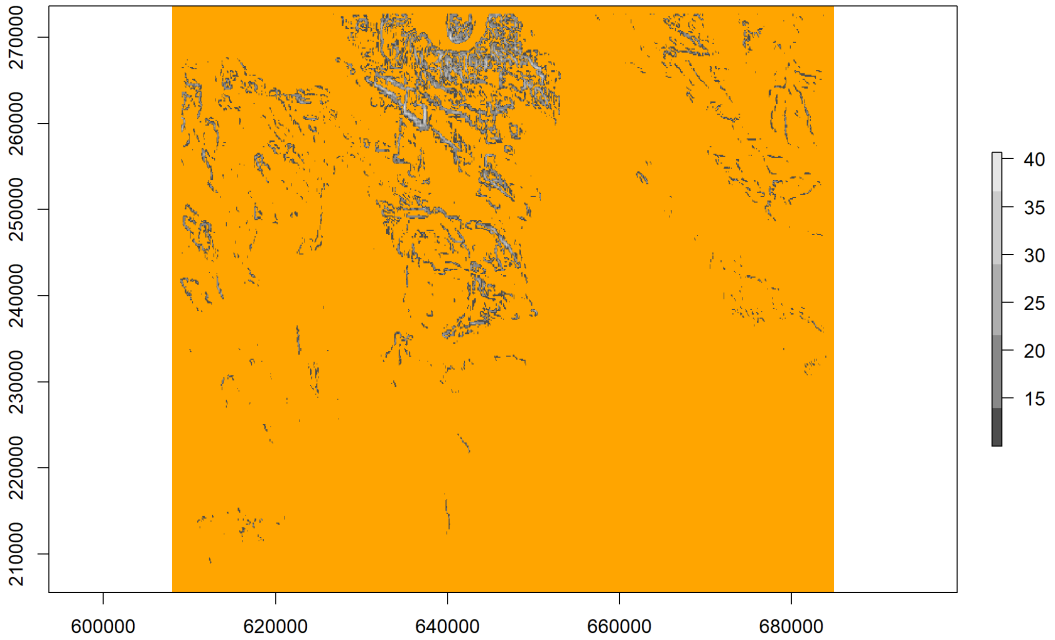
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.659   1.601   2.974   3.942  40.655   29040
```

```
plot(lejtoszog_fok)
```



# Terepjellemzők

```
meredek_lejtok <- lejtoszog_fok
meredek_lejtok[meredek_lejtok[] < 10] <- NA
plot(meredek_lejtok, col = gray.colors(5), colNA = "orange")
```



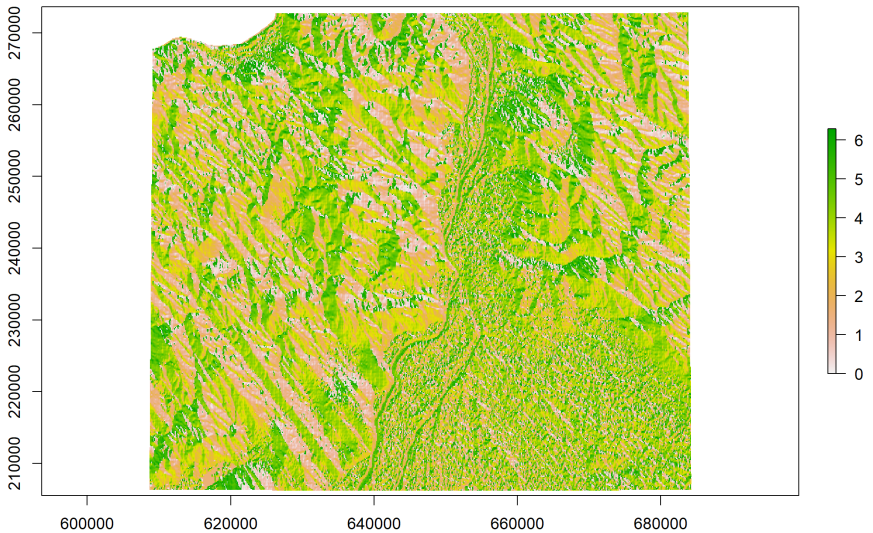
# Terepjellemzők

```
kitettseg_radian <- terrain(x = dem, opt = "aspect", unit
 = "radians")
```

```
summary(kitettseg_radian[])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   1.387   3.142   2.956   4.313   6.283   29040
```

```
plot(kitettseg_radian)
```



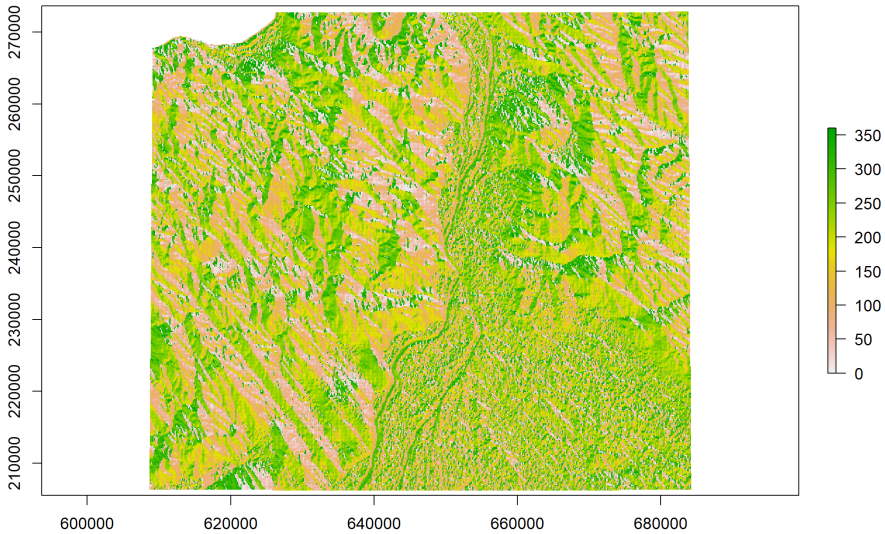
# Terepjellemzők

```
kitettseg_fok <- terrain(x = dem, opt = "aspect", unit =
 "degrees")
```

```
summary(kitettseg_fok[])
```

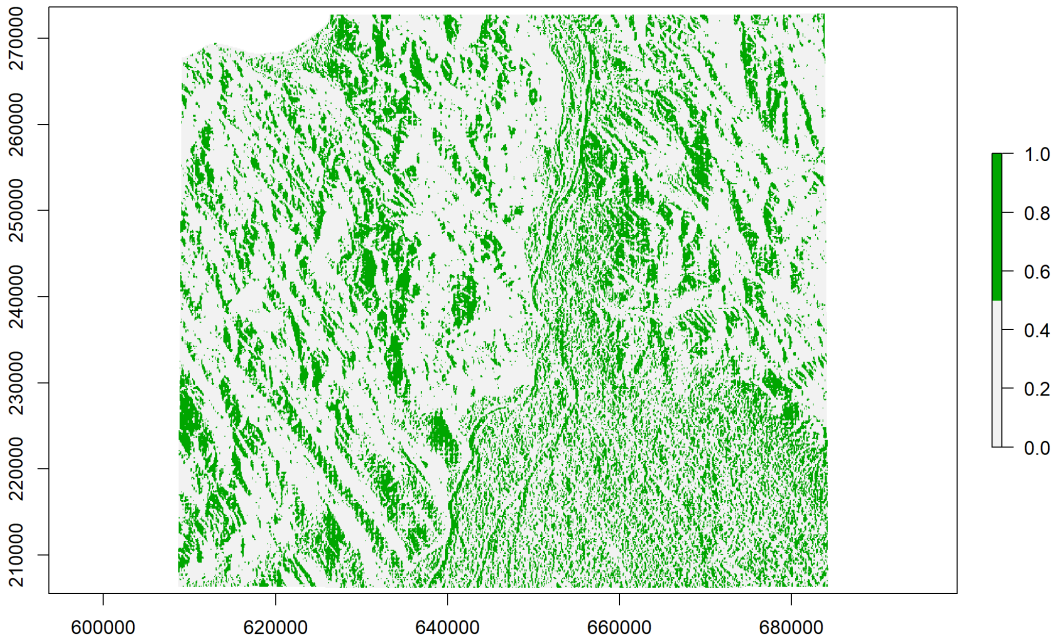
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00   79.49  180.00  169.38  247.11  360.00   29040
```

```
plot(kitettseg_fok)
```



# Terepjellemzők

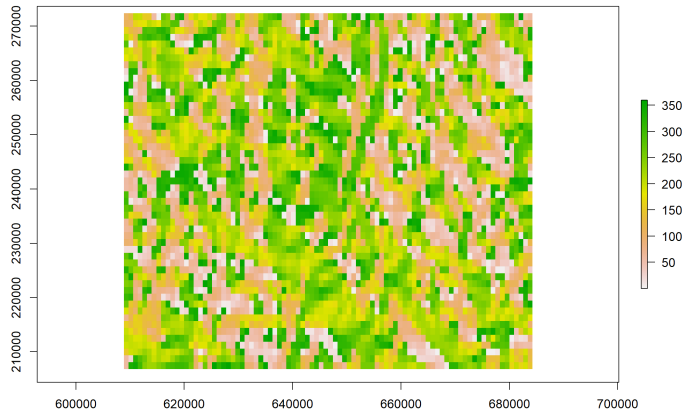
```
nyugati_lejtok <- kitettseg_fok > 225 & kitettseg_fok < 315
plot(nyugati_lejtok, col = rev(terrain.colors(2)))
```

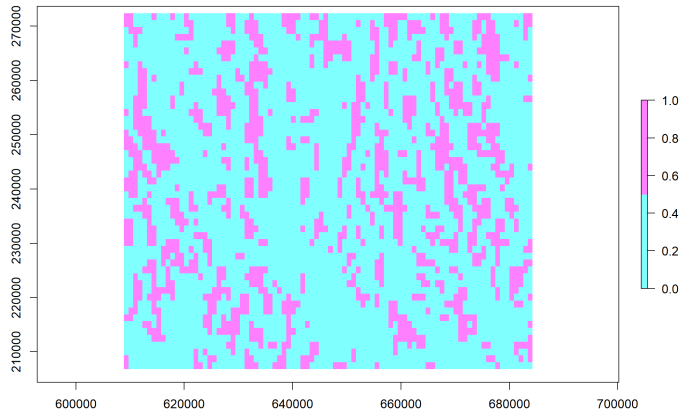



A raszter nem tartalmazhat TRUE/FALSE értékeket, csak 1/0-kat, ezért automatikusan számmá alakul a logikai kifejezés eredménye.

# 5. feladat (házi)

- Aggregáld a felszinhomerseklet nevű rasztert 10×10-es blokkokban, átlagolást alkalmazva, majd jelenítsd meg az eredményt.
- Számítsd ki belőle a hőmérsékleti változás mértékét (lejtőszöget), és jelenítsd meg.
- Számítsd ki a változás irányát (kitettséget) és jelenítsd meg.
- Készíts "logikai" (1/0) rasztert, ami azt jelzi, hogy a cella keleti irányban (45°–135°) hűl-e. Jelenítsd meg két színt alkalmazva.

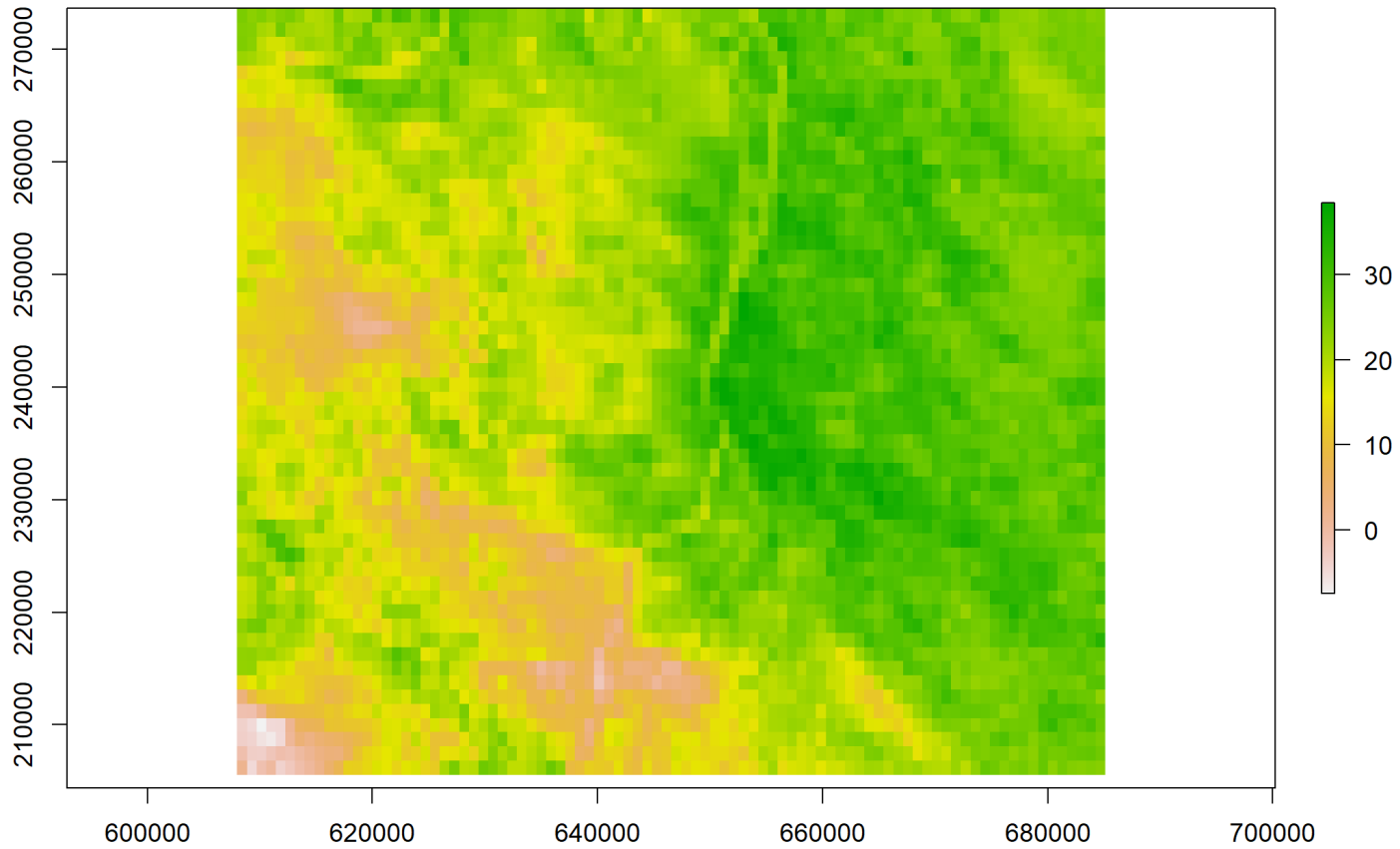


# 5. feladat (házi) – megoldás

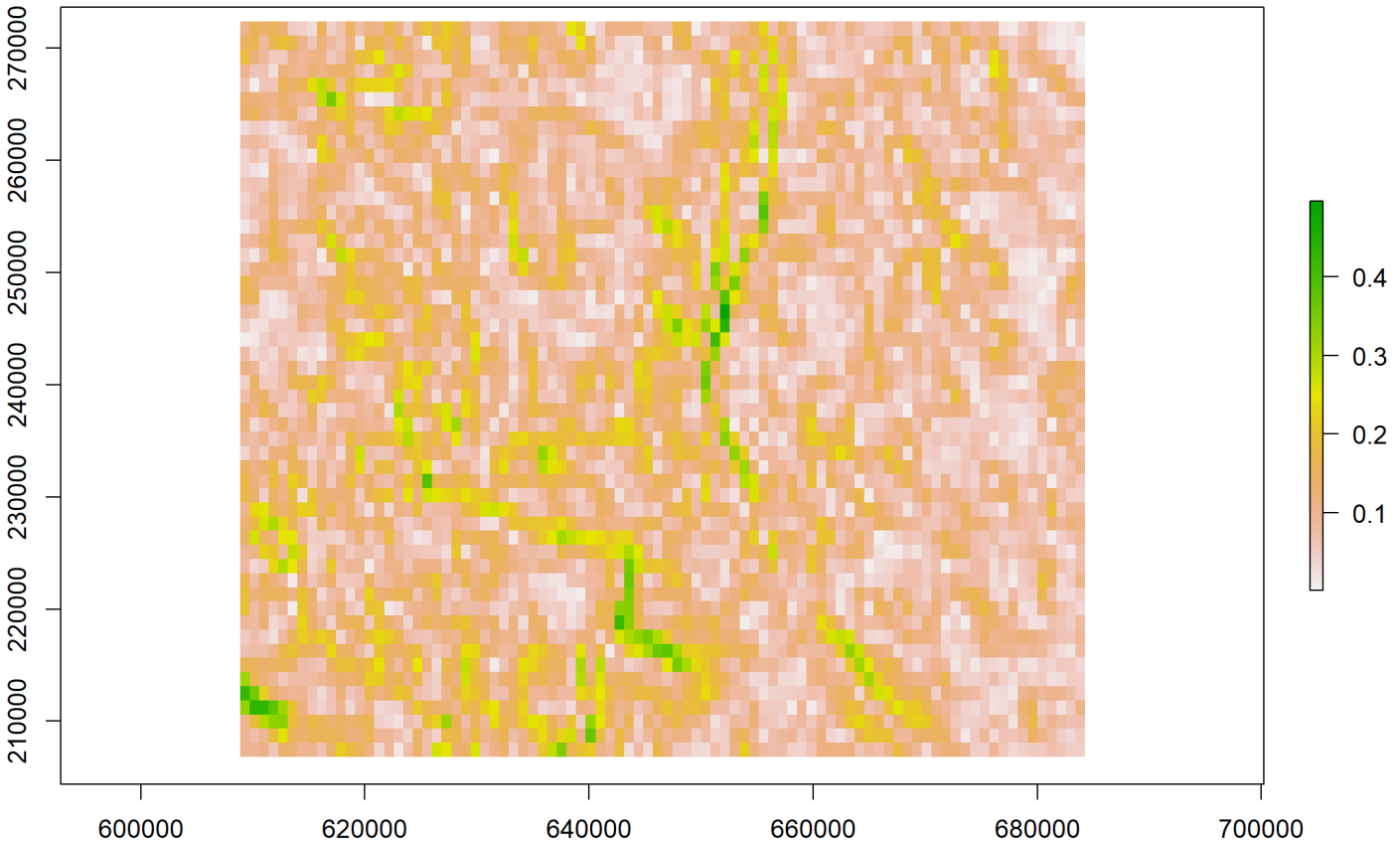
```
homerseklet_durva <- aggregate(x = felszinhomerseklet,
 fact = 10, fun = mean)
```

```
plot(homerseklet_durva)
```



# 5. feladat (házi) – megoldás

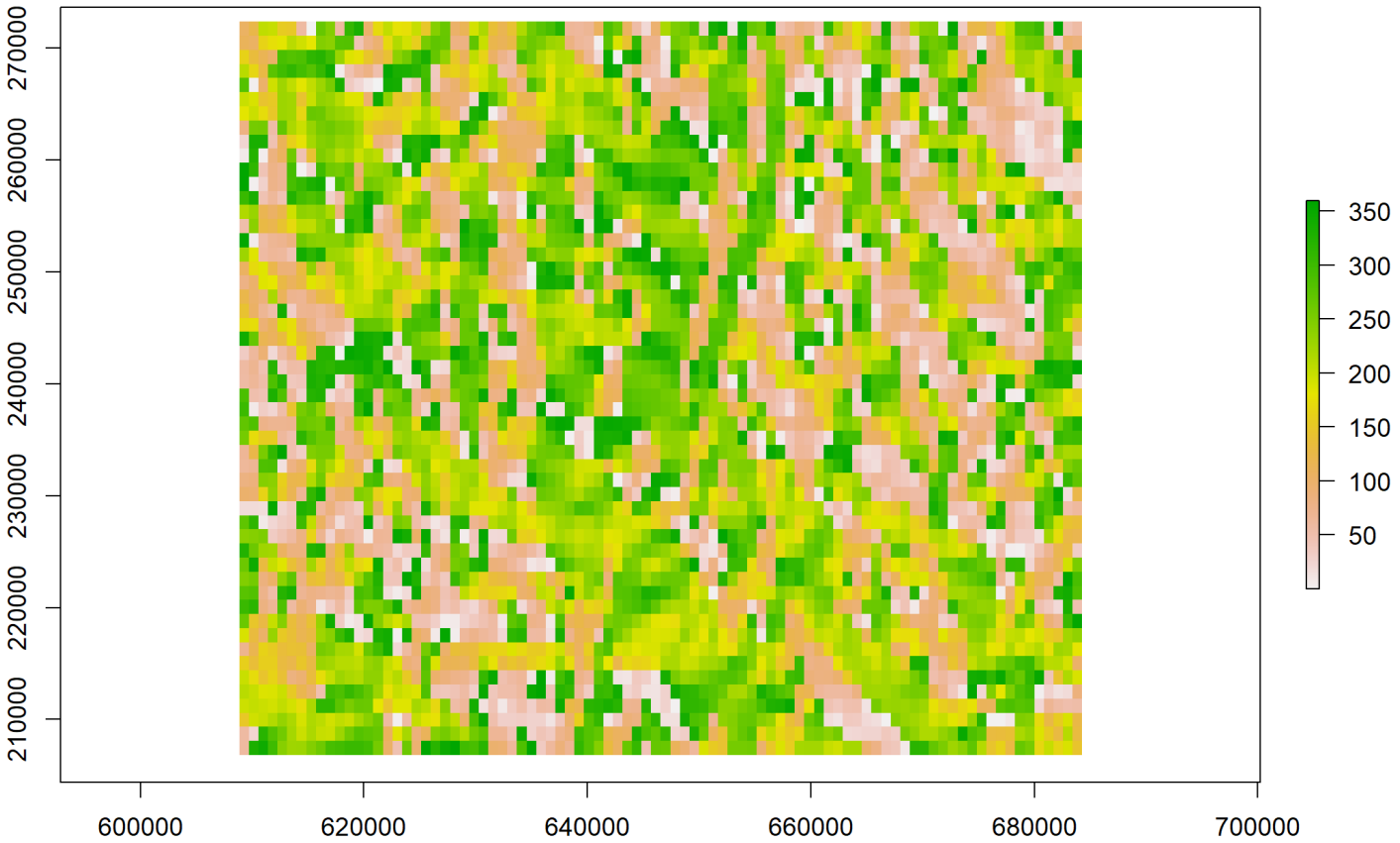
```
valtozas_merteke <- terrain(x = homerseklet_durva, opt =
 "slope", unit = "degrees")
plot(valtozas_merteke)
```



# 5. feladat (házi) – megoldás

```
valtozas_iranya <- terrain(x = homerseklet_durva, opt =
 "aspect", unit = "degrees")
```

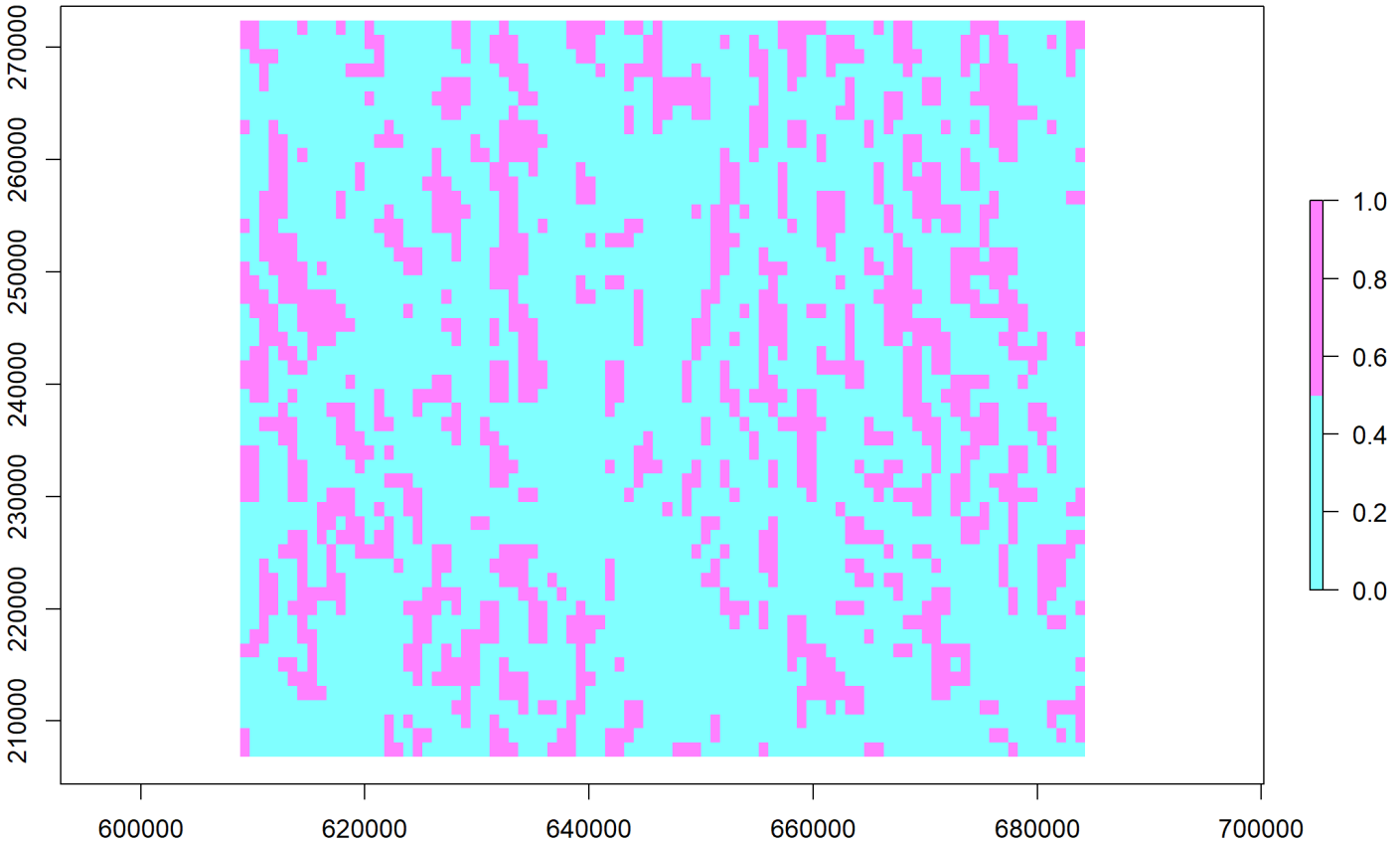
```
plot(valtozas_iranya)
```



# 5. feladat (házi) – megoldás

```
keleti_iranyba_hulnek <- valtozas_iranya > 45 &
 valtozas_iranya < 135
```

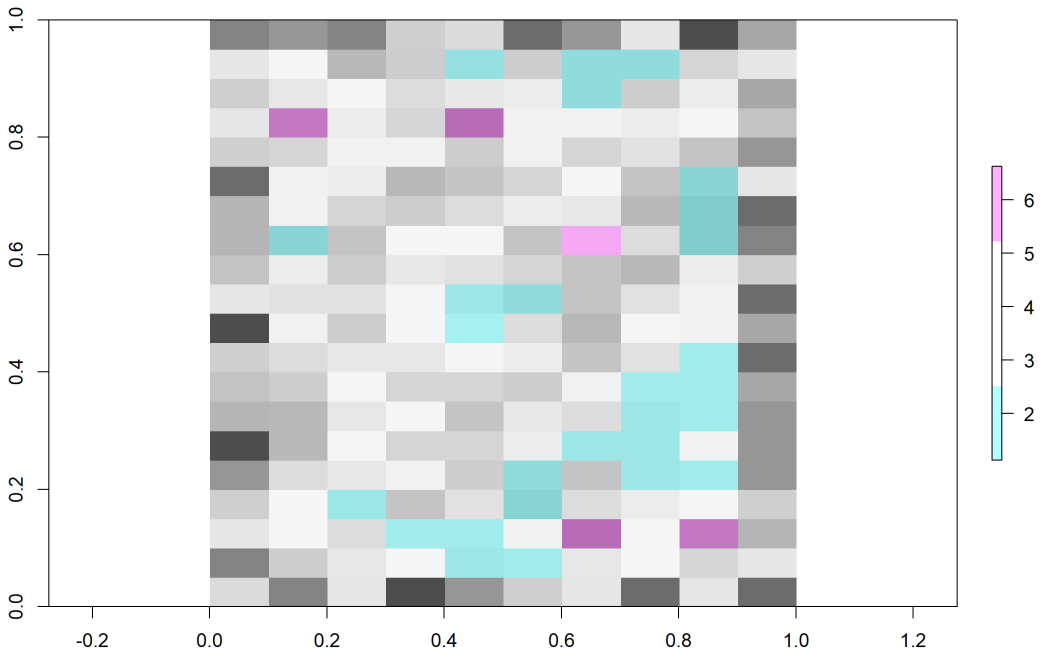
```
plot(keleti_iranyba_hulnek, col = cm.colors(2))
```



# Terepjellemzők

Terrain Ruggedness Index:

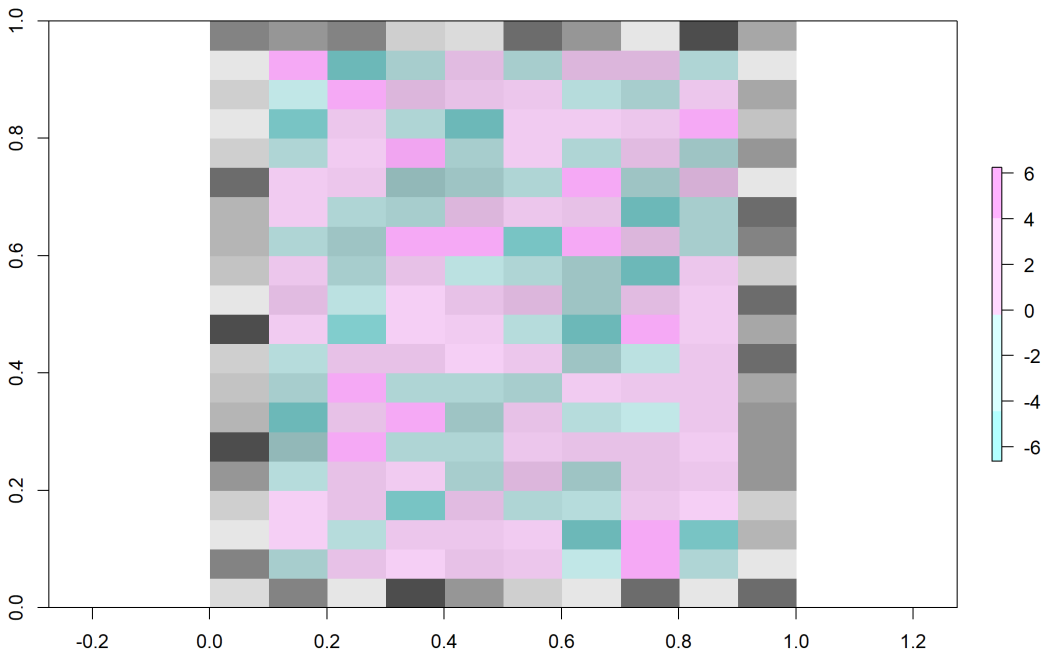
```
plot(szamraszter, col = gray.colors(100), legend = FALSE)
valtozatossag <- terrain(x = szamraszter, opt = "tri")
plot(valtozatossag, add = TRUE, col = cm.colors(3), alpha
 = 0.6)
```



# Terepjellemzők

Topographic Position Index:

```
topografiai_helyzet <- terrain(x = szamraszter, opt = "tpi")
plot(szamraszter, col = gray.colors(100), legend = FALSE)
plot(topografiai_helyzet, add = TRUE, col = cm.colors(4),
 alpha = 0.6)
```



# Domborzatárnyékolás

## Domborzatárnyékolás

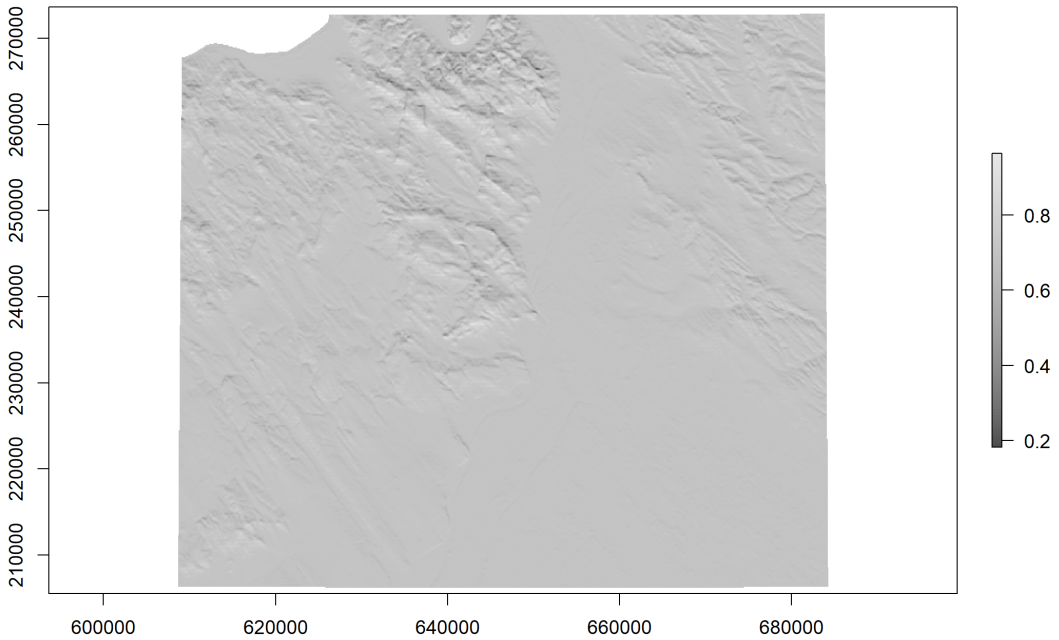
- elsősorban grafikai felhasználásra képezzük
- kiemeli, plasztikusabbá teszi a domborzatot
- áttetszőséget kell alkalmazni

## `hillShade(slope, aspect, angle = 45, direction = 0)`

- `slope`: lejtőszögraszter radiánban
- `aspect`: kitettségraszter radiánban
- `angle`: napfény beesési szöge a vízszínteshez képest
- `direction`: a napfény érkezési iránya az északhoz képest
- kartográfiai sztenderd szerint északról süt a nap...

# Domborzatárnyékolás

```
arnyekolas <- hillShade(slope = lejtoszog_radian, aspect =
 kitettseg_radian, angle = 45, direction = 180)
plot(arnyekolas, col = gray.colors(100))
```

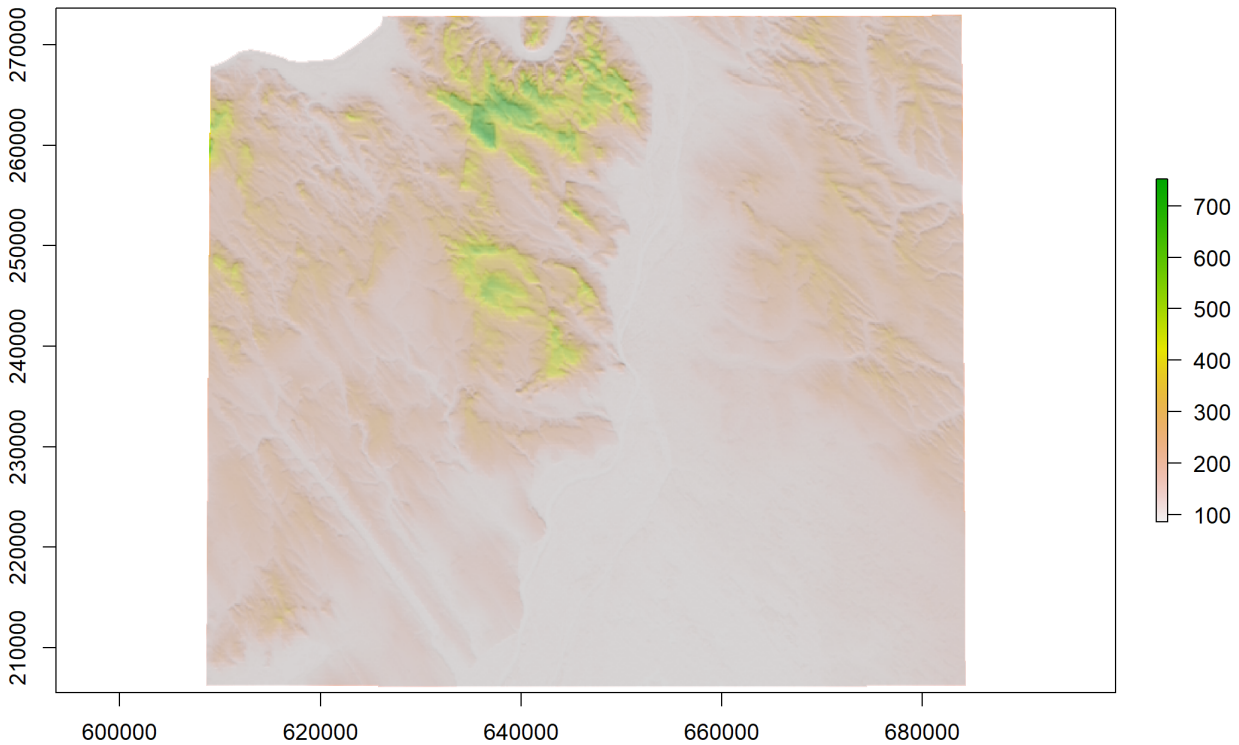



Önmagában nincs sok értelme megjeleníteni.

# Domborzatárnyékolás

Árnyék hozzáadása utólag, áttetszőséggel:

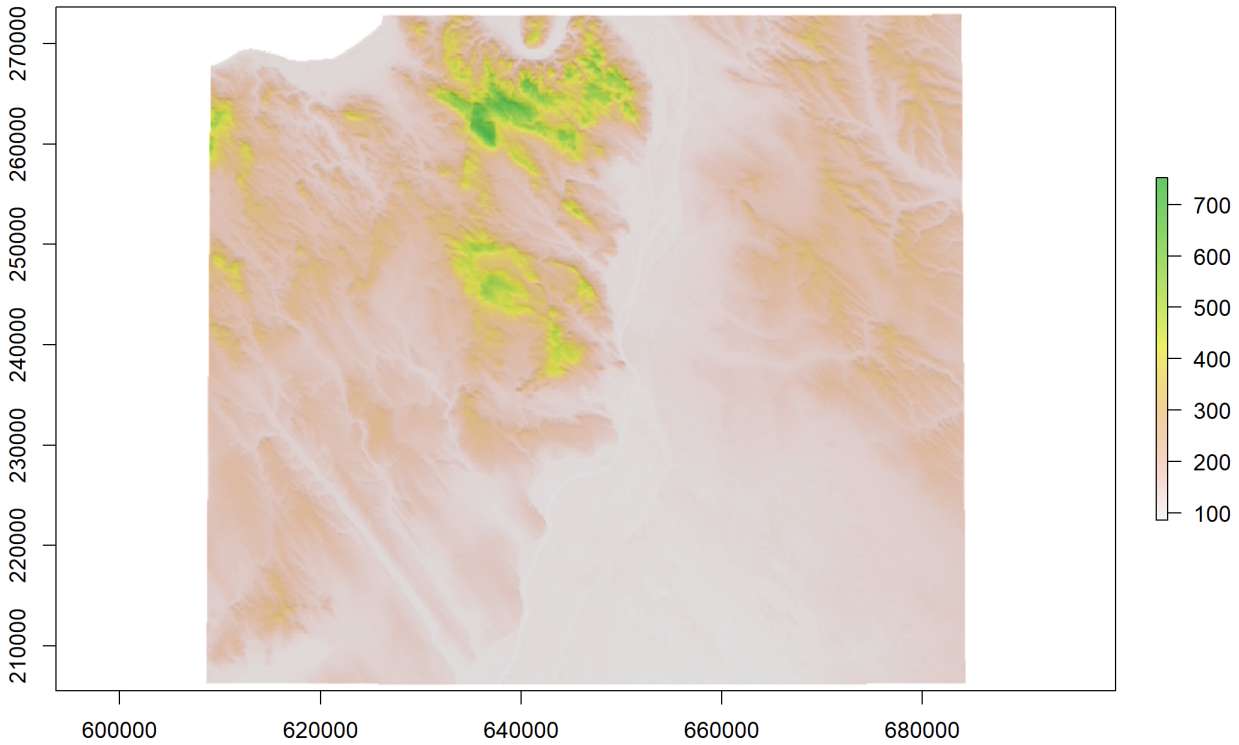
```
plot(dem)
plot(arnyekolas, add = TRUE, col = gray.colors(100), alpha
 = 0.6, legend = FALSE)
```



# Domborzatárnyékolás

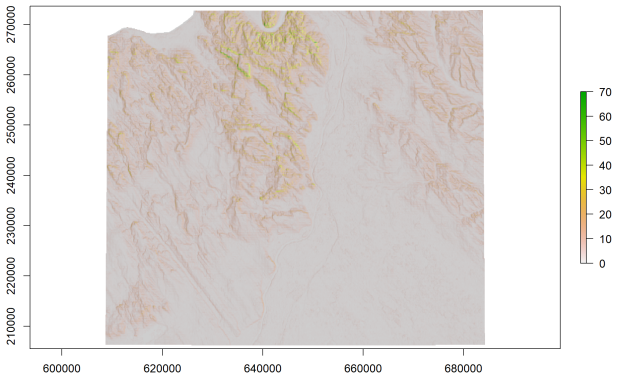
Vagy fordítva:

```
plot(arnyekolas, col = gray.colors(100), legend = FALSE)
plot(dem, alpha = 0.6, add = TRUE)
```



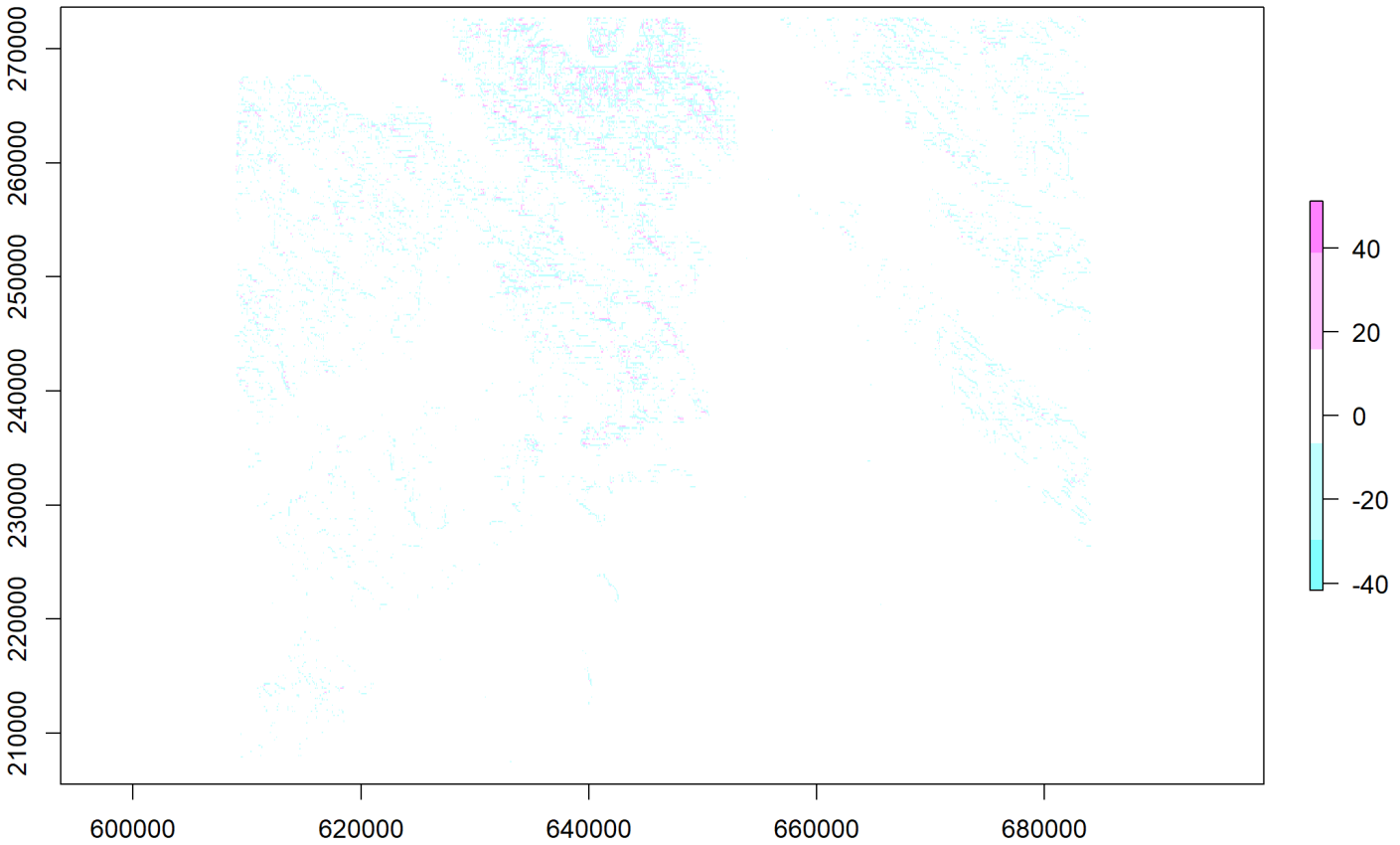
# 6. feladat (házi)

- Számítsd ki a “dem” nevű domborzatmodell celláinak topográfiai helyzetét (TPI). Jelenítsd meg úgy, hogy a `cm.colors` palettáról 5 színt használsz.
- Számítsd ki a domborzatmodell változatosságát (TRI), és jelenítsd meg.
- Készíts kora reggeli domborzatárnyékolást, vagyis alacsony beesési szöggel, és keleties iránnyal dolgozz.
- Az előző ábrához 60%-os átlátszatlansággal add hozzá a domborzatárnyékolást jelmagyarázat nélkül, a szürke színpalettáról (`gray.colors`) 100 színt választva.



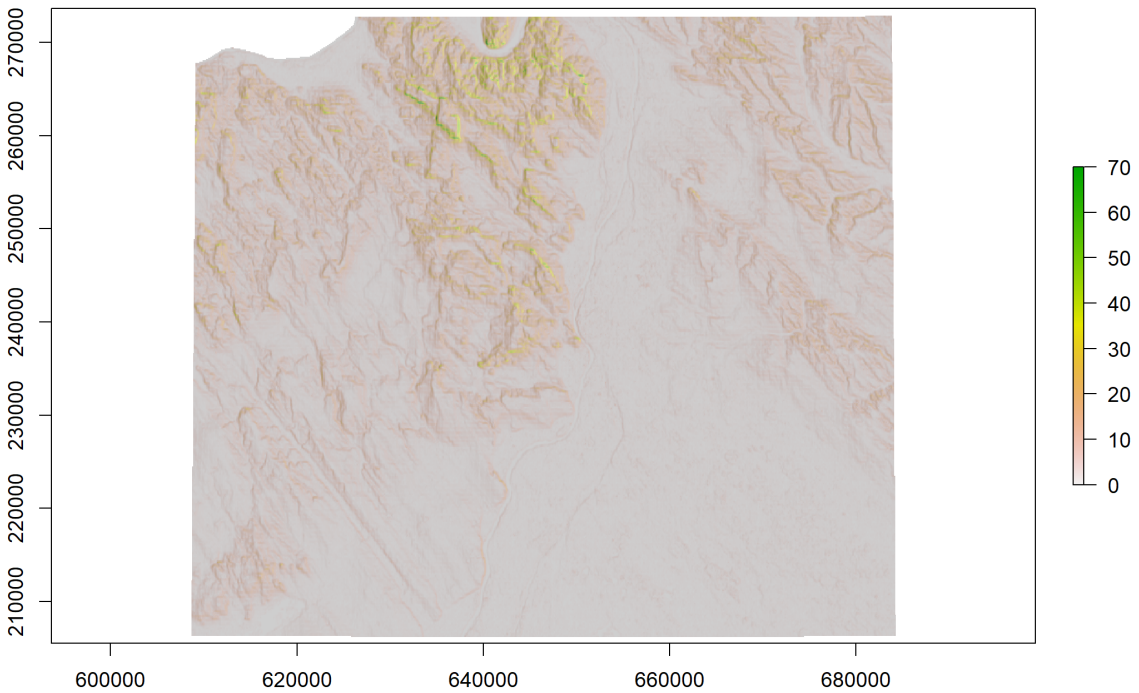
# 6. feladat (házi) – megoldás

```
dem_topografiai_helyzet <- terrain(x = dem, opt = "tpi")
plot(dem_topografiai_helyzet, col = cm.colors(5))
```



# 6. feladat (házi) – megoldás

```
dem_valtozatossag <- terrain(x = dem, opt = "tri")
plot(dem_valtozatossag)
reggeli_arnyekolas <- hillShade(slope = lejtoszog_radian,
 aspect = kitettseg_radian, angle = 10, direction = 100)
plot(reggeli_arnyekolas, add = TRUE, col =
 gray.colors(100), alpha = 0.6, legend = FALSE)
```



Section 4

# Távolságszámítás

# Távolságszámítás

- három módszer
- eredmény: egy, a bemeneti raszteréhez hasonló tulajdonságokkal bíró raszter, ami minden cellában távolságértéket tartalmaz

## `distance(x)`

- ismeretlen értékű cellák távolsága az ismertektől
- `x`: egyrétegű raszter, amely tartalmaz ismeretlen értékű (`NA`) cellákat

## `gridDistance(x, origin)`

- adott értékű celláktól vett távolság
- `x`: egyrétegű raszter
- `origin`: egy vagy több számérték. Az ilyen értékeket tartalmazó celláktól méri a távolságot
- nem légvonalban számol, hanem szomszédos cellákon lépkedve

# Távolságszámítás

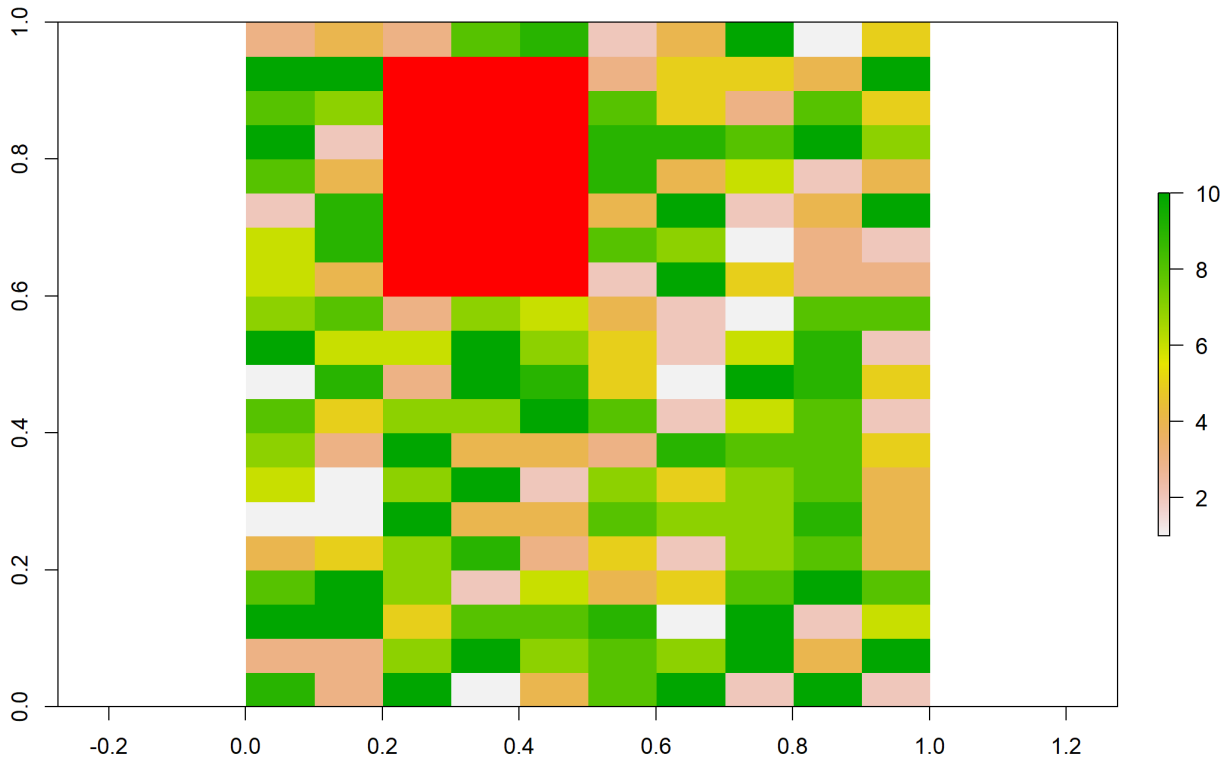
## `distanceFromPoints(object, xy)`

- ponttól vett távolság
- `object`: egyrétegű raszter
- `xy`: `POINT` típusú Simple Features
- minden cellának kiszámolja a legközelebbi ponttól vett távolságát

# Távolságszámítás

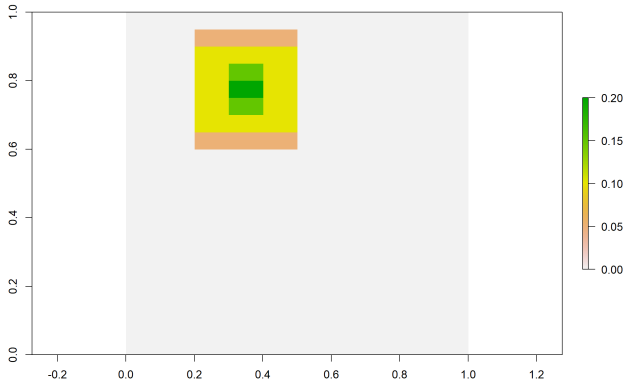
Töröljük néhány cellából az értékeket!

```
szamraszter[2:8, 3:5] <- NA
plot(szamraszter, colNA = "red")
```

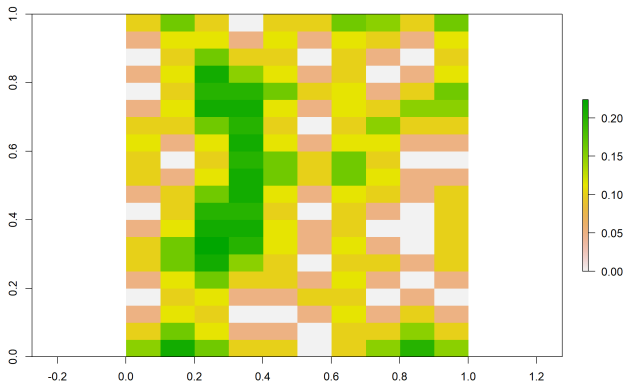


# Távolságszámítás

```
plot(distance(x = szamraszter))
```



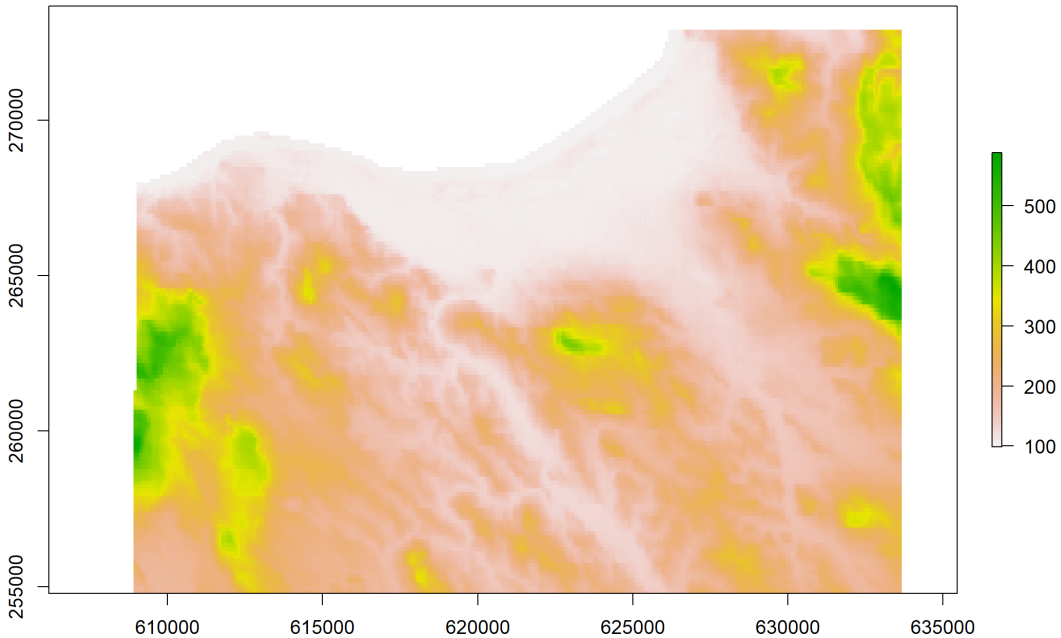

```
plot(gridDistance(x = szamraszter, origin = 8))
```



# Távolságszámítás

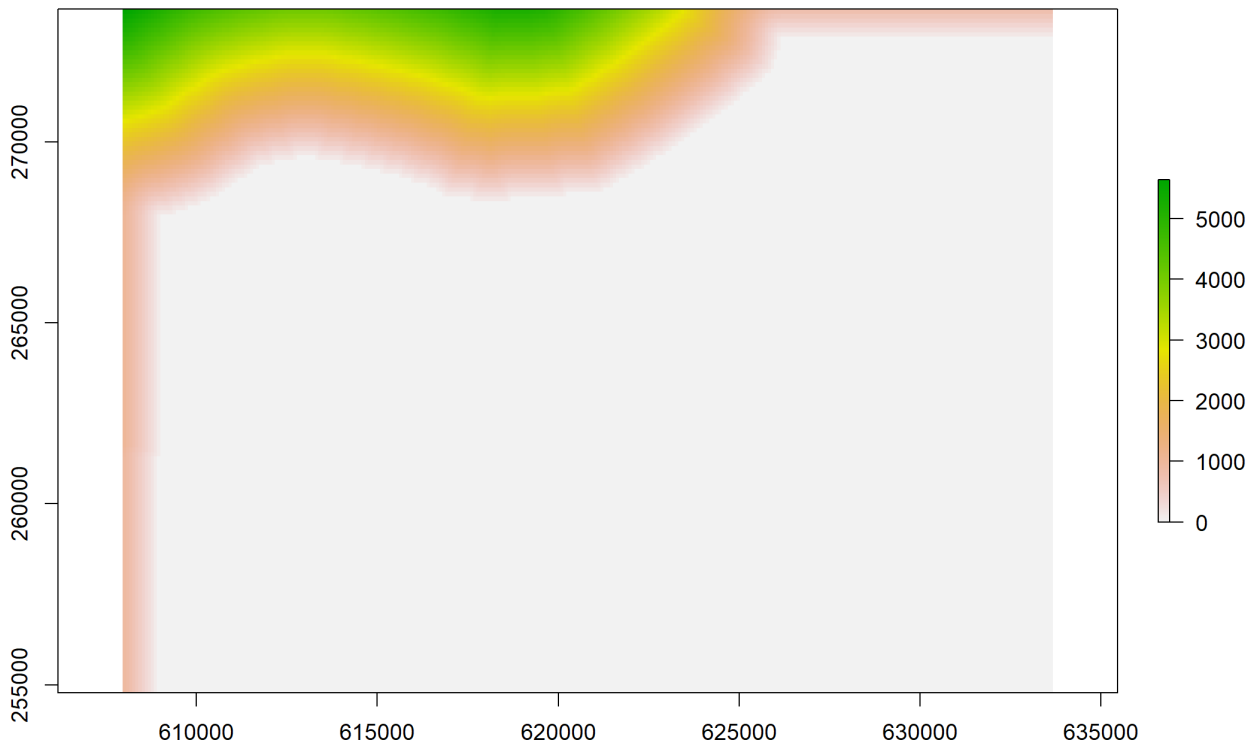
Nézzük meg ugyanezeket valós adatokra!

```
esztergom <- dem[1:150, 1:300, drop = FALSE]
plot(esztergom)
```



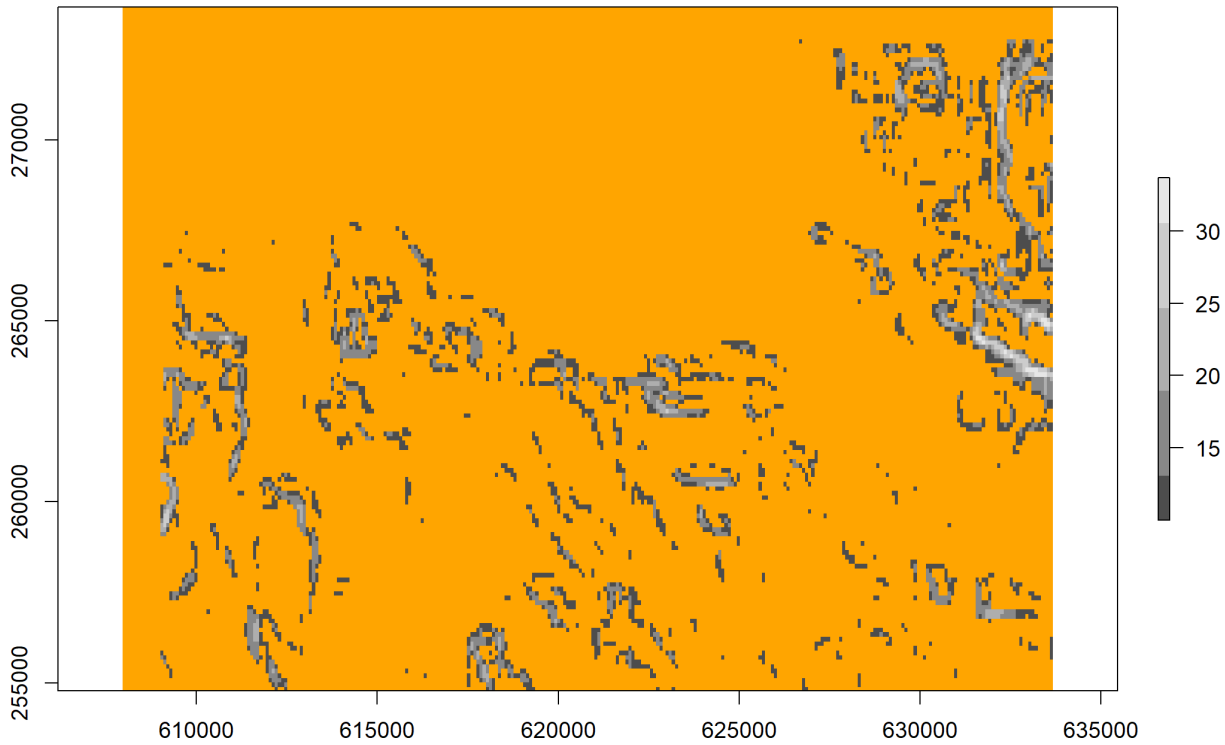
# Távolságszámítás

```
tavolsag_ismert_cellatol <- distance(x = esztergom)
plot(tavolsag_ismert_cellatol)
```



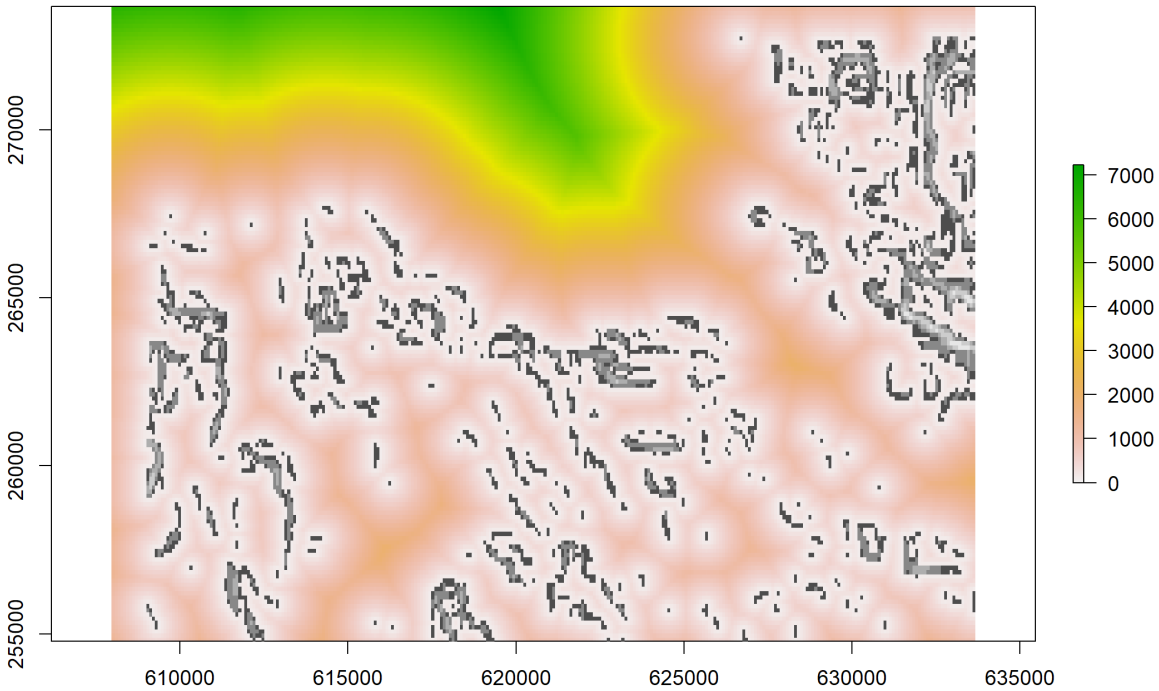
# Távolságszámítás

```
meredek_lejtok_kivagat <- meredek_lejtok[1:150, 1:300,
 drop = FALSE]
plot(meredek_lejtok_kivagat, col = gray.colors(5), colNA =
 "orange")
```



# Távolságszámítás

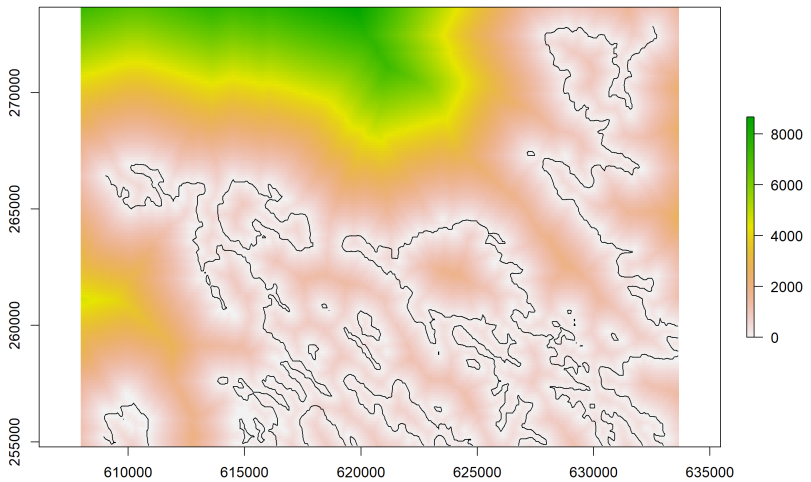
```
tavolsag_ismert_cellatol <- distance(x =
 meredek_lejtok_kivagat)
plot(tavolsag_ismert_cellatol)
plot(meredek_lejtok_kivagat, col = gray.colors(5), add =
 TRUE, legend = FALSE)
```



# Távolságszámítás

A körülbelül 200 m tszf. magasságú celláktól mérjünk távolságot!

```
tavolsag_200mes_szinttol <- gridDistance(x = esztergom,
 origin = 195:205)
plot(tavolsag_200mes_szinttol)
contour(x = esztergom, levels = 200, drawlabels = FALSE,
 add = TRUE)
```



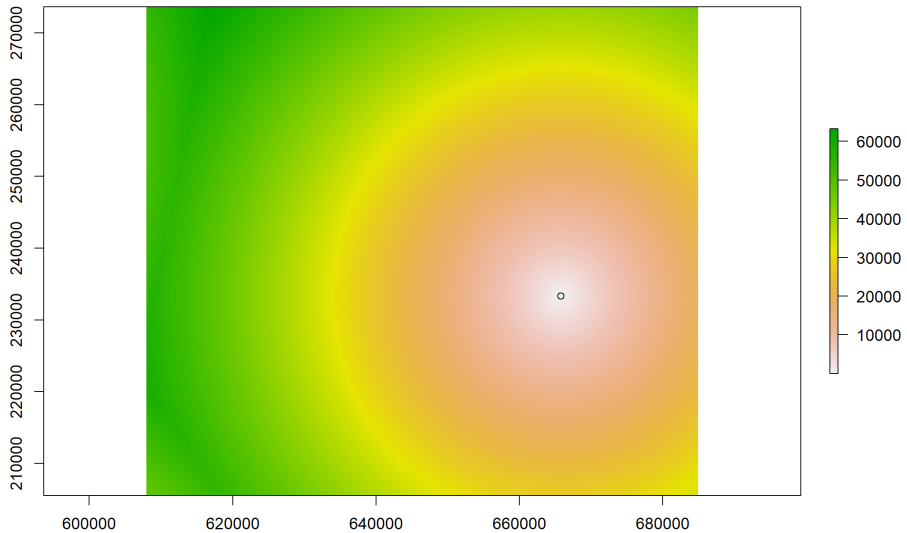

A `contour()` függvénnyel szintvonalakat ábrázolhatunk.

# Távolságszámítás

```
library(sf)
load("repterek.RData")
```

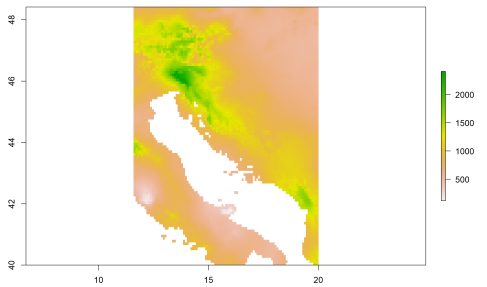
```
tavolsagok_repterektol <- distanceFromPoints(object = dem,
 xy = repterek)
```

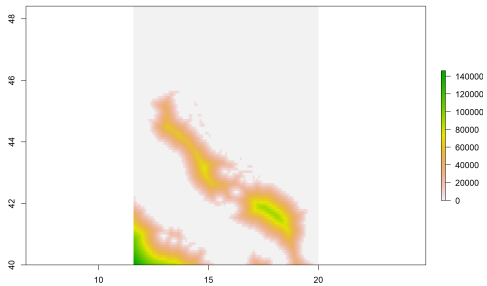
```
plot(tavolsagok_repterektol)
plot(st_geometry(repterek), add = TRUE)
```

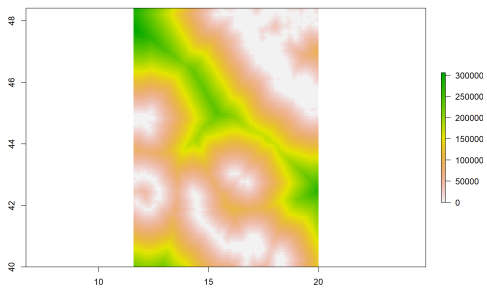


# 7. feladat (házi)

- Vágd ki a “csapadek” nevű raszterből Horvátország környékét (500–600. sorok, 2300–2400. oszlopok), és jelenítsd meg.
- Számold ki e kivágott raszteren minden ismeretlen értékű (tengerre eső) cellának a távolságát az ismert csapadékú celláktól.
- Ezt is jelenítsd meg.
- Számold ki minden cella távolságát az 550–650 mm csapadékú celláktól, majd jelenítsd meg.



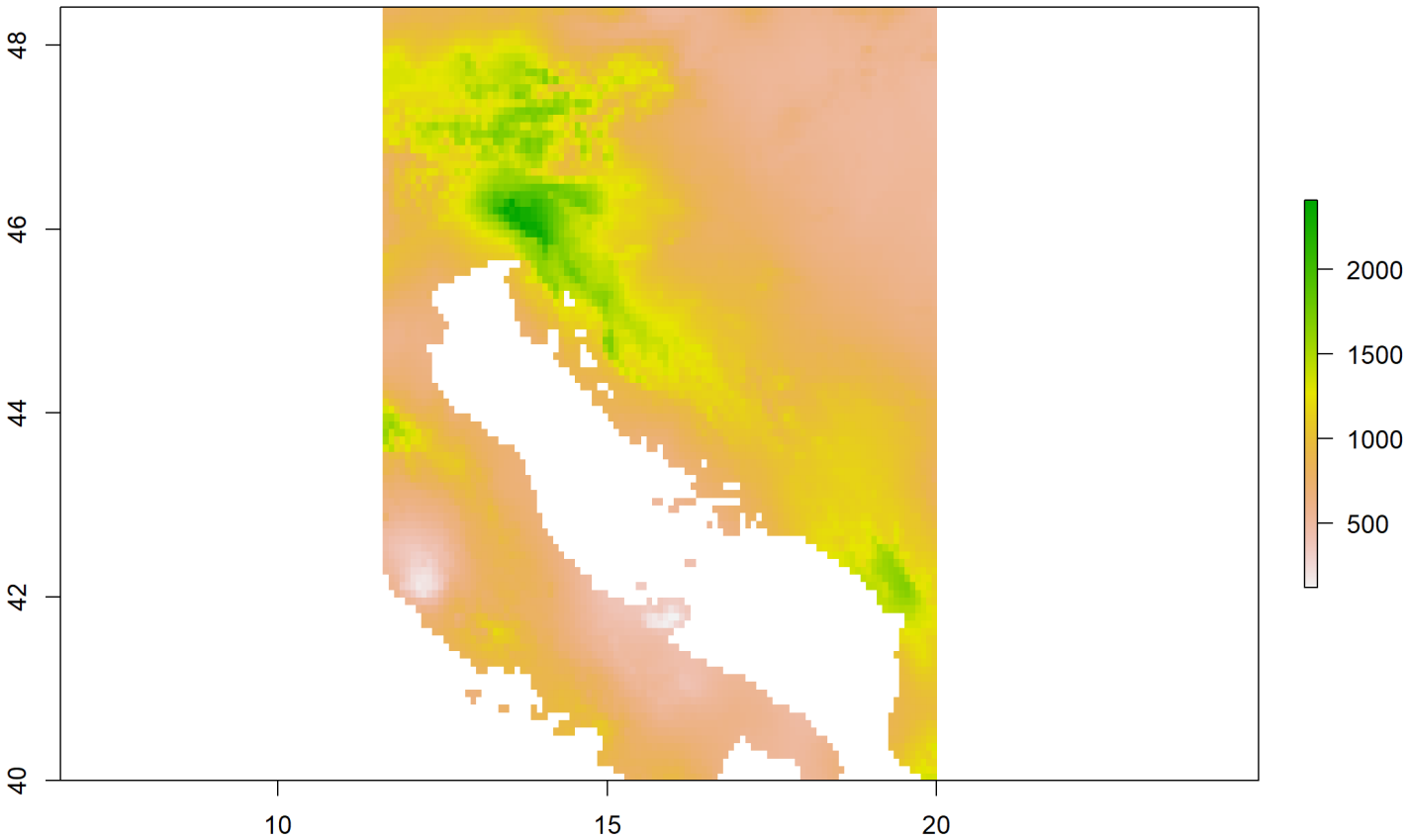



# 7. feladat (házi) – megoldás

```
horvat_csapadek <- csapadek[500:600, 2300:2400, drop =
 FALSE]
```

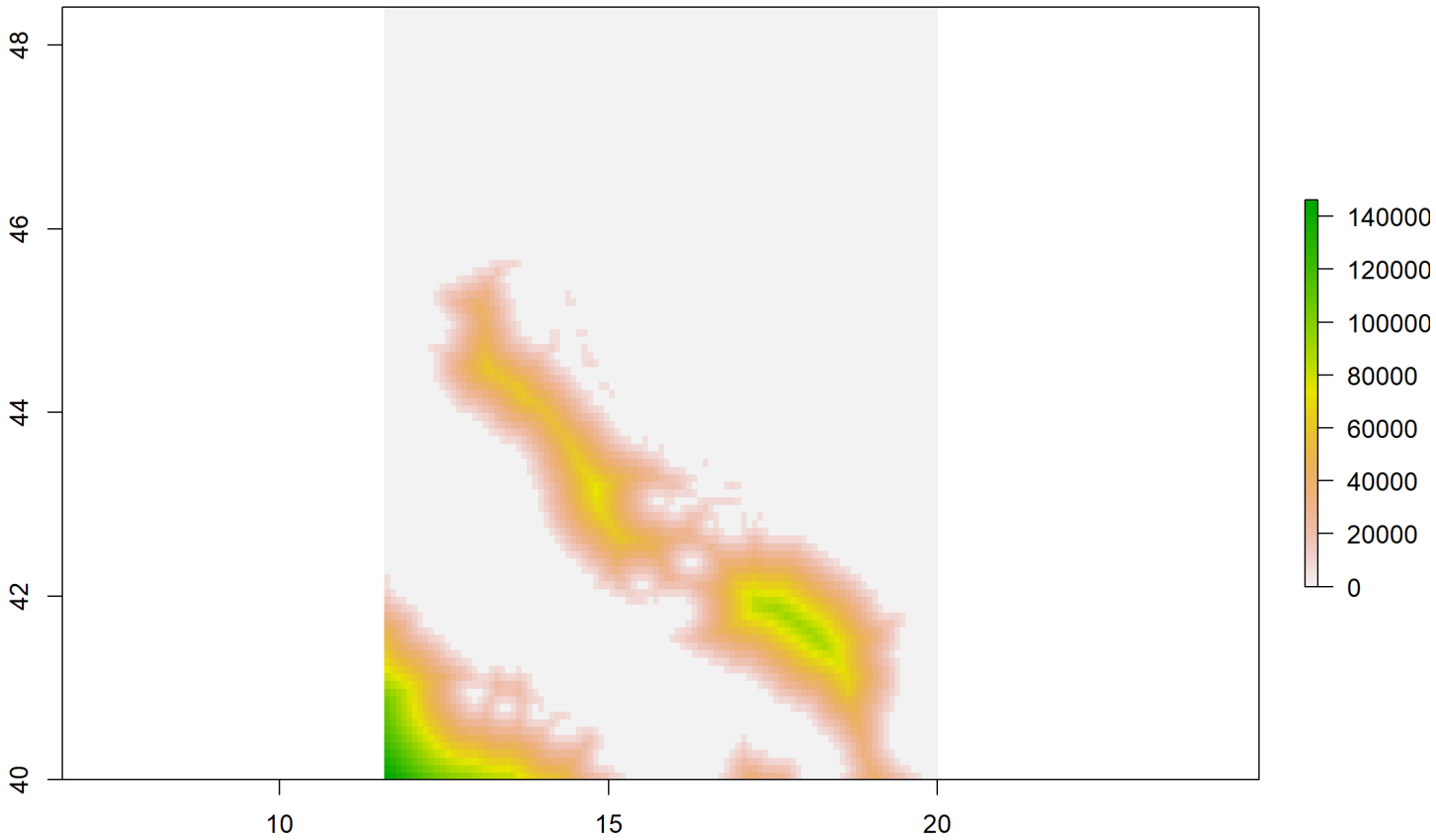
```
plot(horvat_csapadek)
```



# 7. feladat (házi) – megoldás

```
tavolsag_ismert_cellatol <- distance(x = horvat_csapadek)
```

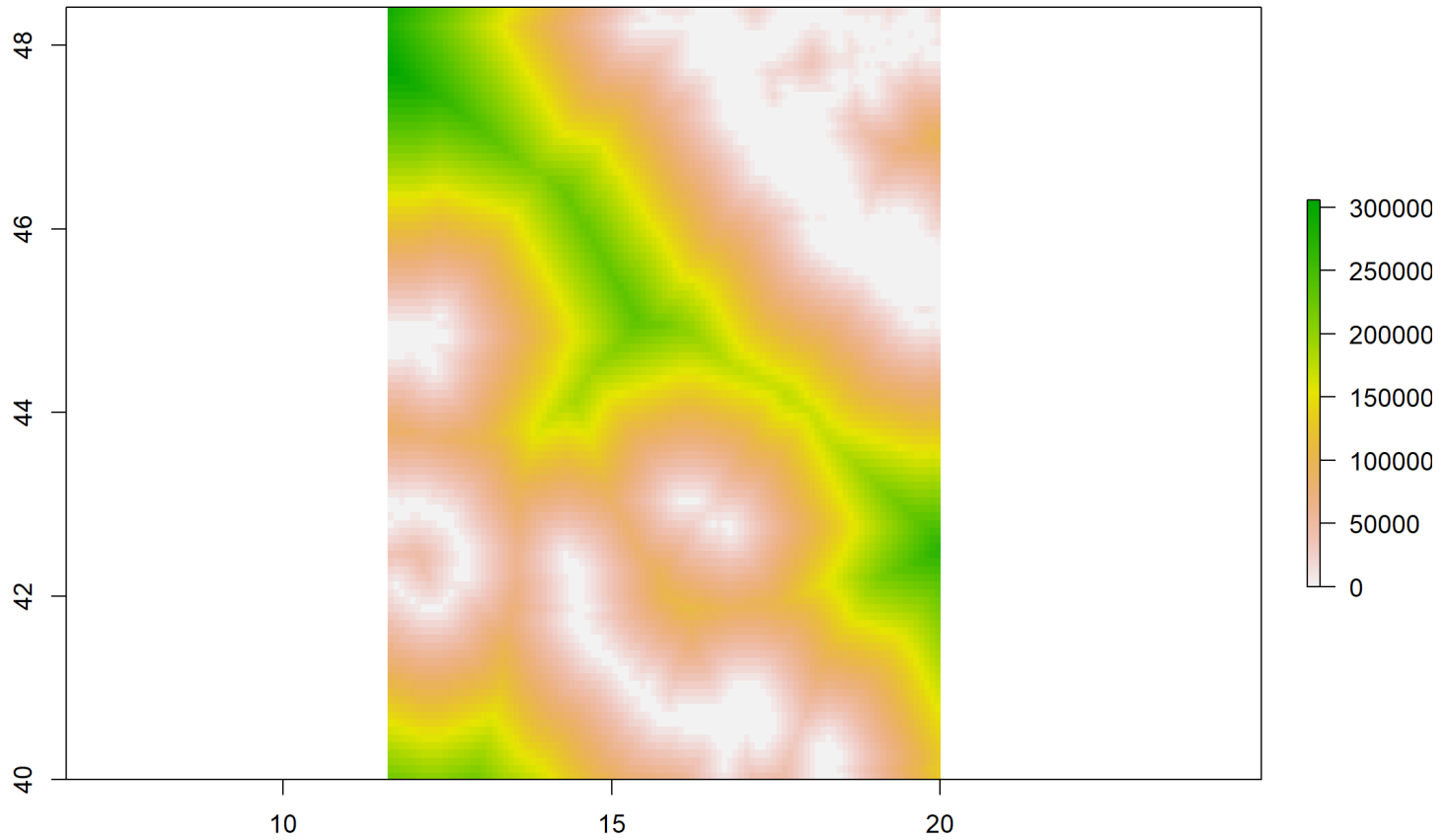
```
plot(tavolsag_ismert_cellatol)
```



# 7. feladat (házi) – megoldás

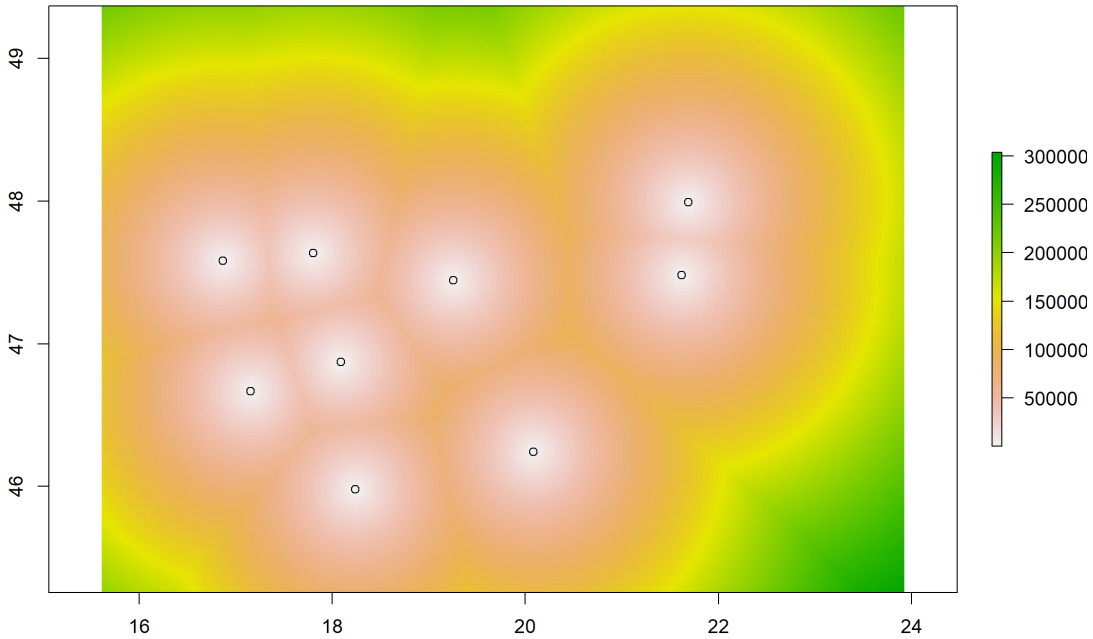
```
tavolsag_600mmtol <- gridDistance(x = horvat_csapadek,
 origin = 550:650)
```

```
plot(tavolsag_600mmtol)
```



# 8. feladat (órai)

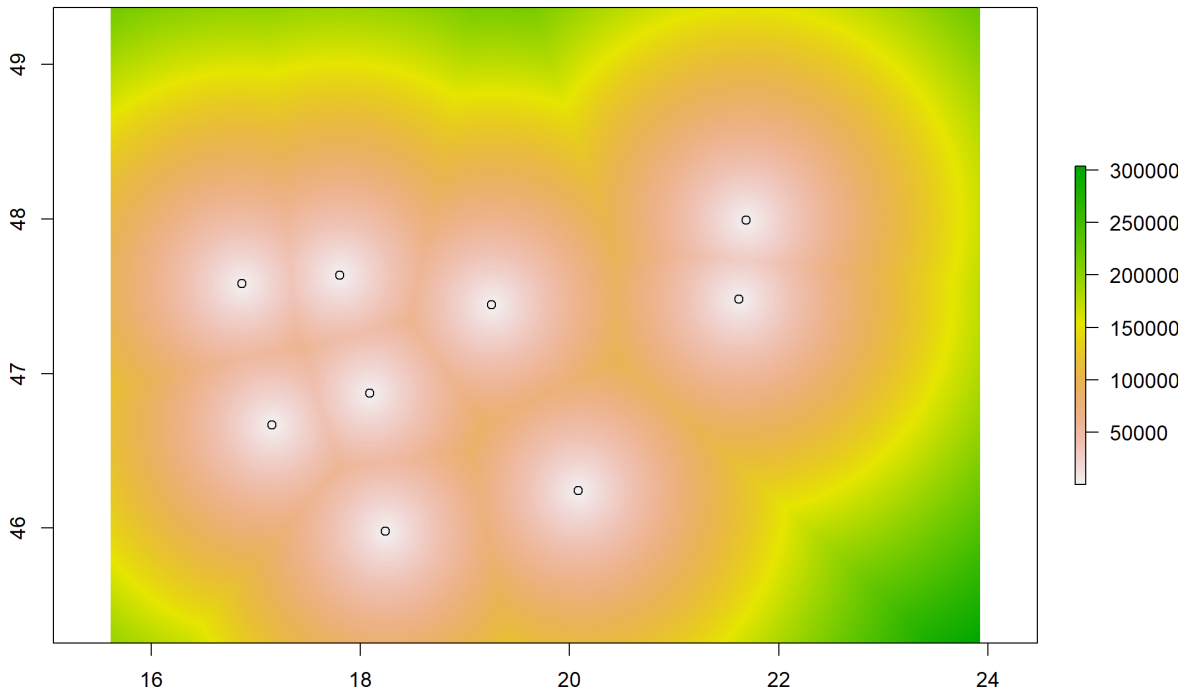
- Hozz létre a “repterek” nevű Simple Featuresből másolatot, amit WGS-84-be (EPSG: 4326) vetítesz.
- Számold ki a “felszinboritas” nevű raszter minden cellájára, hogy milyen távol esik a legközelebbi repülőtértől.
- Jelenítsd meg az eredményrasztert, és add hozzá a repterek geometriáját ellenőrzésképpen.



# 8. feladat (órai) – megoldás

```
repterek_wgs <- st_transform(x = repterek, crs = 4326)
tavolsagok_repterektol <- distanceFromPoints(object =
 felszinboritas, xy = repterek_wgs)
```

```
plot(tavolsagok_repterektol)
plot(st_geometry(repterek_wgs), add = TRUE)
```



Section 5

# Vektorrá átalakítás

# Raszter vektorrá átalakítása

## Raszter vektorrá átalakítása

- 2 lehetőség
- `rasterToPolygons()`: poligonná alakít
- `rasterToPoints()`: ponttá (vagy mátrixszá) alakít

## `rasterToPolygons(x, dissolve = FALSE)`

- poligonná alakít minden cellát
- `x`: az átalakítandó raszter
- `dissolve`: összevonja-e egy poligonná a szomszédos, ugyanolyan értéket tartalmazó cellákat?
- sokáig eltarthat, főleg, ha össze is vonunk
- az eredmény az `sp` csomag szerinti `SpatialPolygonsDataFrame`
- ezt könnyen Simple Features-zé alakíthatjuk az `st_as_sf()` függvénnyel
- az eredeti értékek egy "layer" nevű oszlopba kerülnek

# Raszter vektorrá átalakítása

`rasterToPoints(x, spatial = FALSE)`

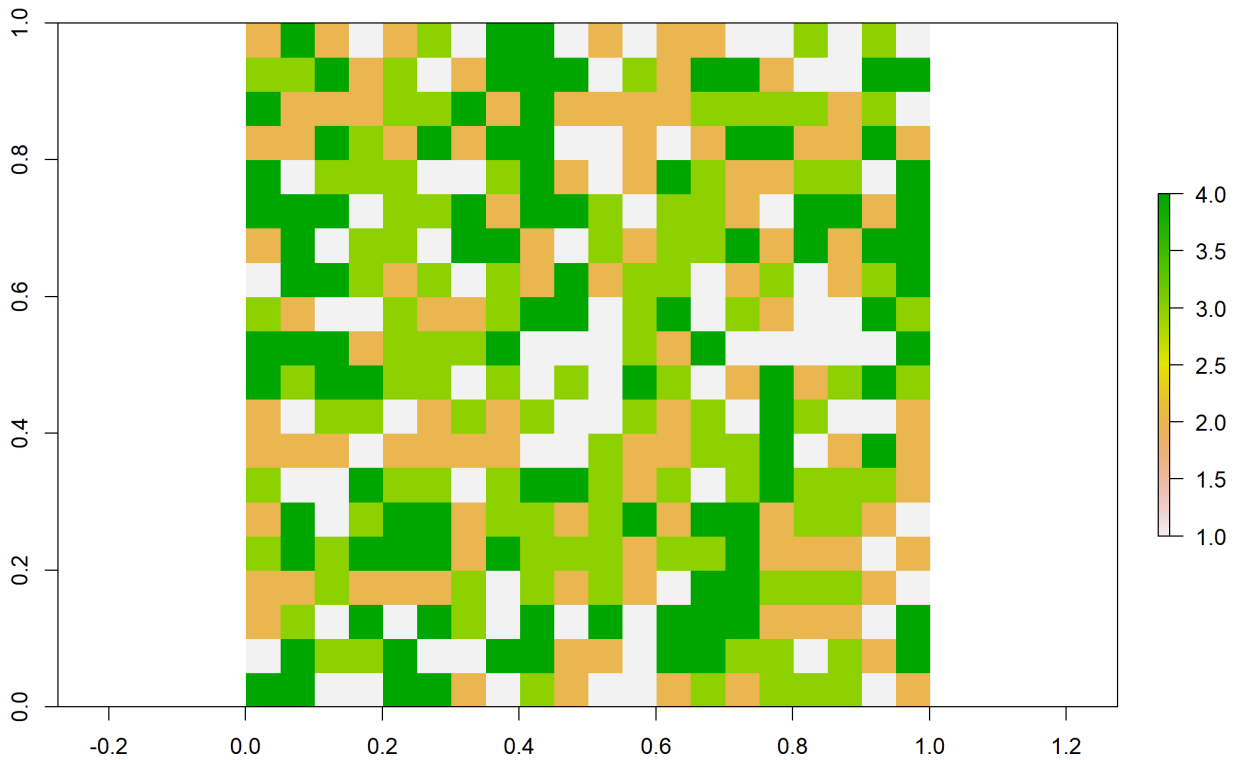
- ponttá vagy mátrixsorrá alakít minden cellát
- `x`: az átalakítandó raszter
- `spatial`: térinformatikai ponthalmazt szeretnénk-e eredményként?
- `spatial = FALSE`: egy háromoszlopos mátrixot ad eredményül (2 koordináta + 1 érték)
- `spatial = TRUE`: az eredmény az `sp` csomag szerinti `SpatialPointsDataFrame`
- ezt könnyen Simple Features-zé alakíthatjuk az `st_as_sf()` függvénnyel

# Raszter poligonná átalakítása

```
set.seed(12345)
szamok2 <- sample(x = 1:4, size = 20 * 20, replace = TRUE)
szammatrix2 <- matrix(data = szamok2, ncol = 20, nrow = 20)
szamraszter2 <- raster(x = szammatrix2)
```

# Raszter poligonná átalakítása

```
plot(szamraszter2)
```



# Raszter poligonná átalakítása

```
poligonkent_kulon <- rasterToPolygons(x = szamraszter2)
class(poligonkent_kulon)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

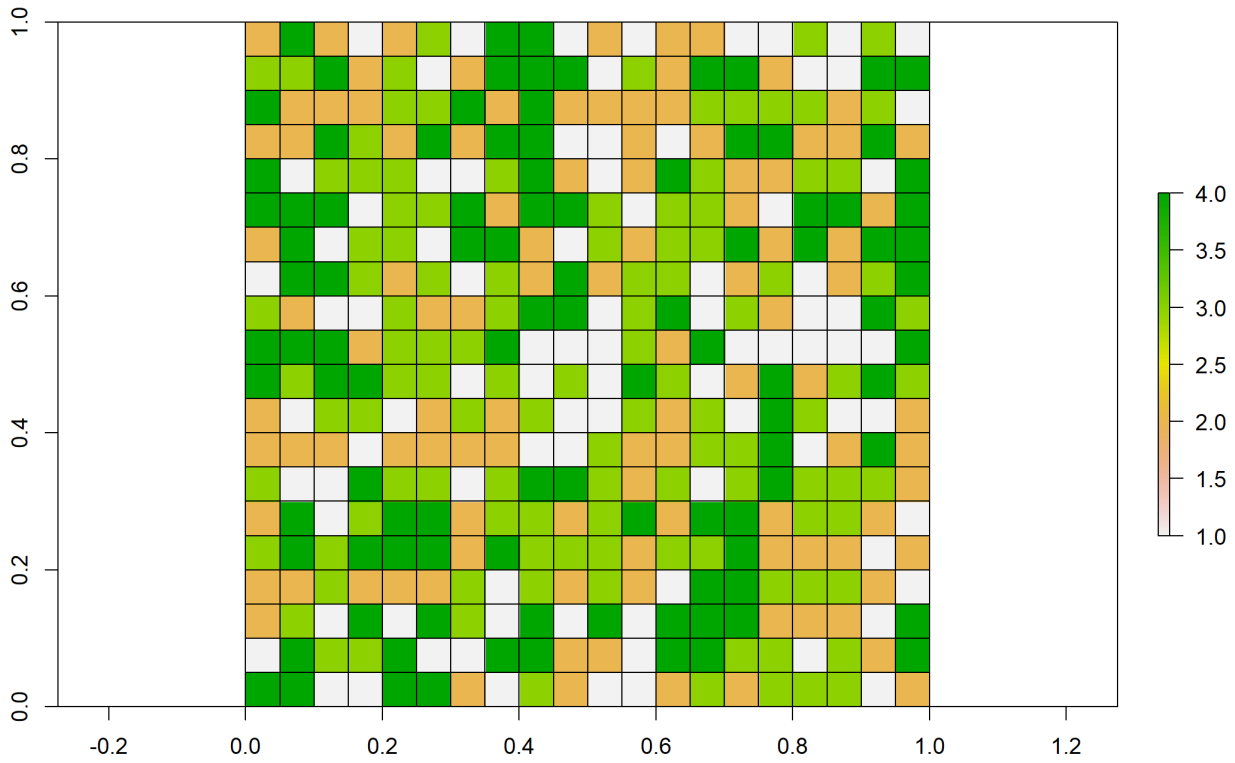
```
poligonkent_kulon <- st_as_sf(poligonkent_kulon)
class(poligonkent_kulon)
```

```
[1] "sf"         "data.frame"
```

# Raszter poligonná átalakítása

Összevonás nélkül (`dissolve = FALSE`, ez az alapértelmezett):

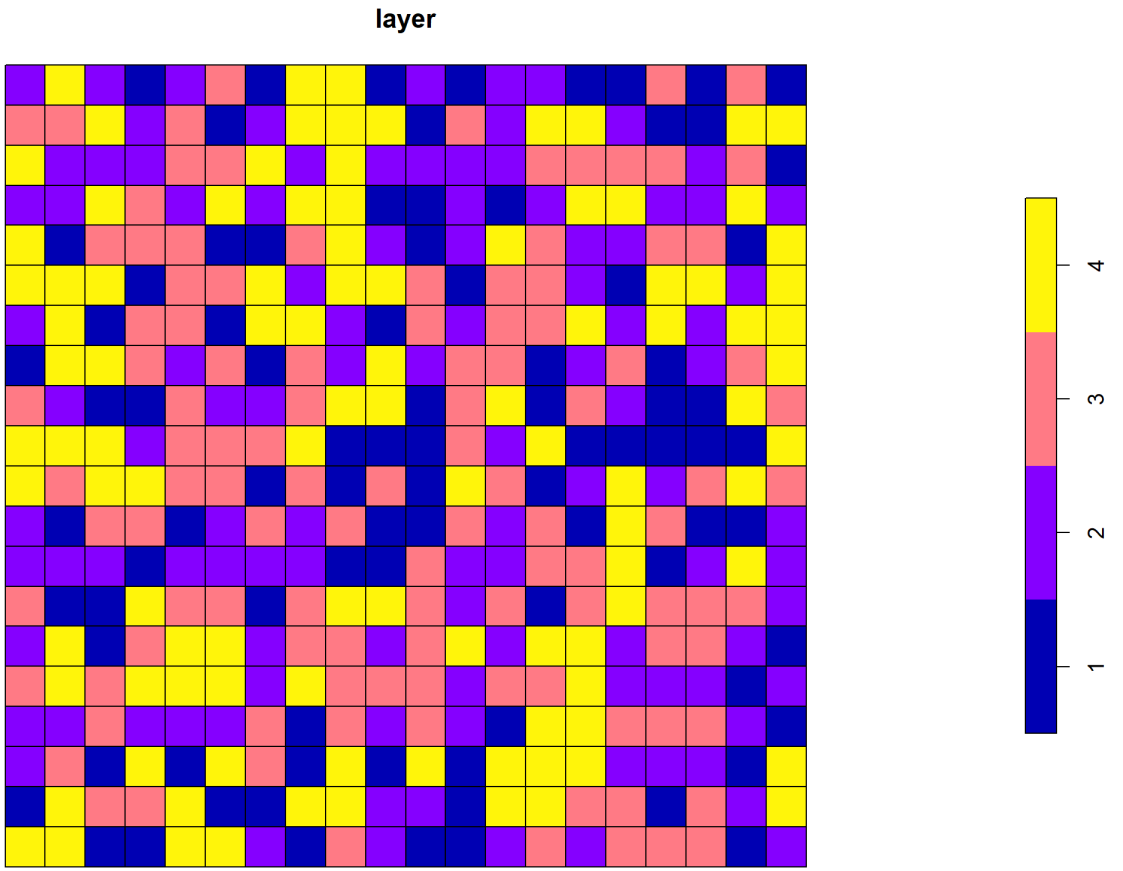
```
plot(st_geometry(poligonkent_kulon), add = TRUE)
```



# Raszter poligonná átalakítása

A “layer” nevű oszlop tartalmazza a volt cellaértékeket.

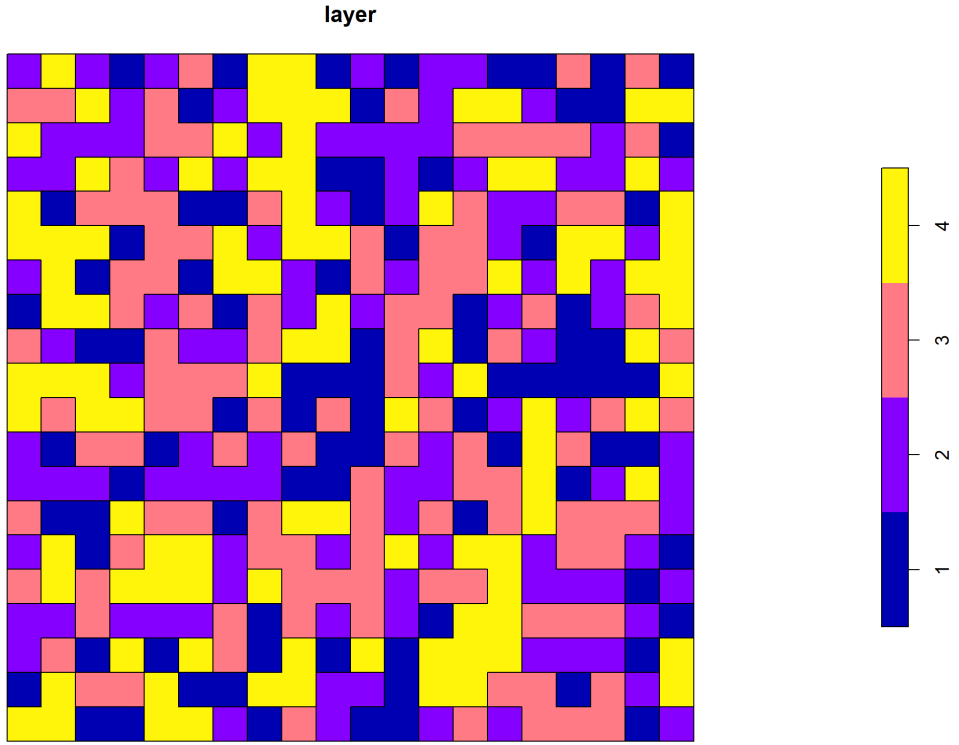
```
plot(poligonkent_kulon)
```



# Raszter poligonná átalakítása

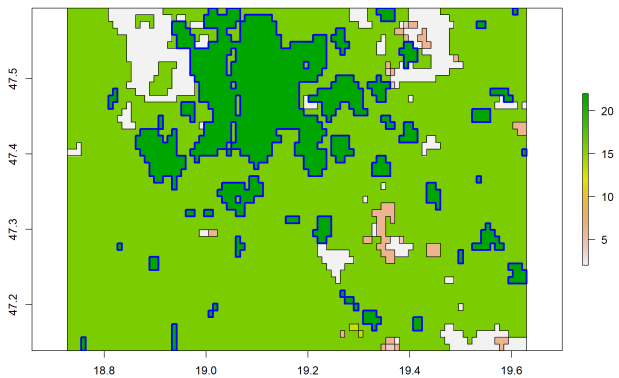
Összevonással (tetrisz...):

```
poligonkent_osszevonva <- st_as_sf(rasterToPolygons(x =
 szamraszter2, dissolve = TRUE))
plot(poligonkent_osszevonva)
```



# 9. feladat (házi)

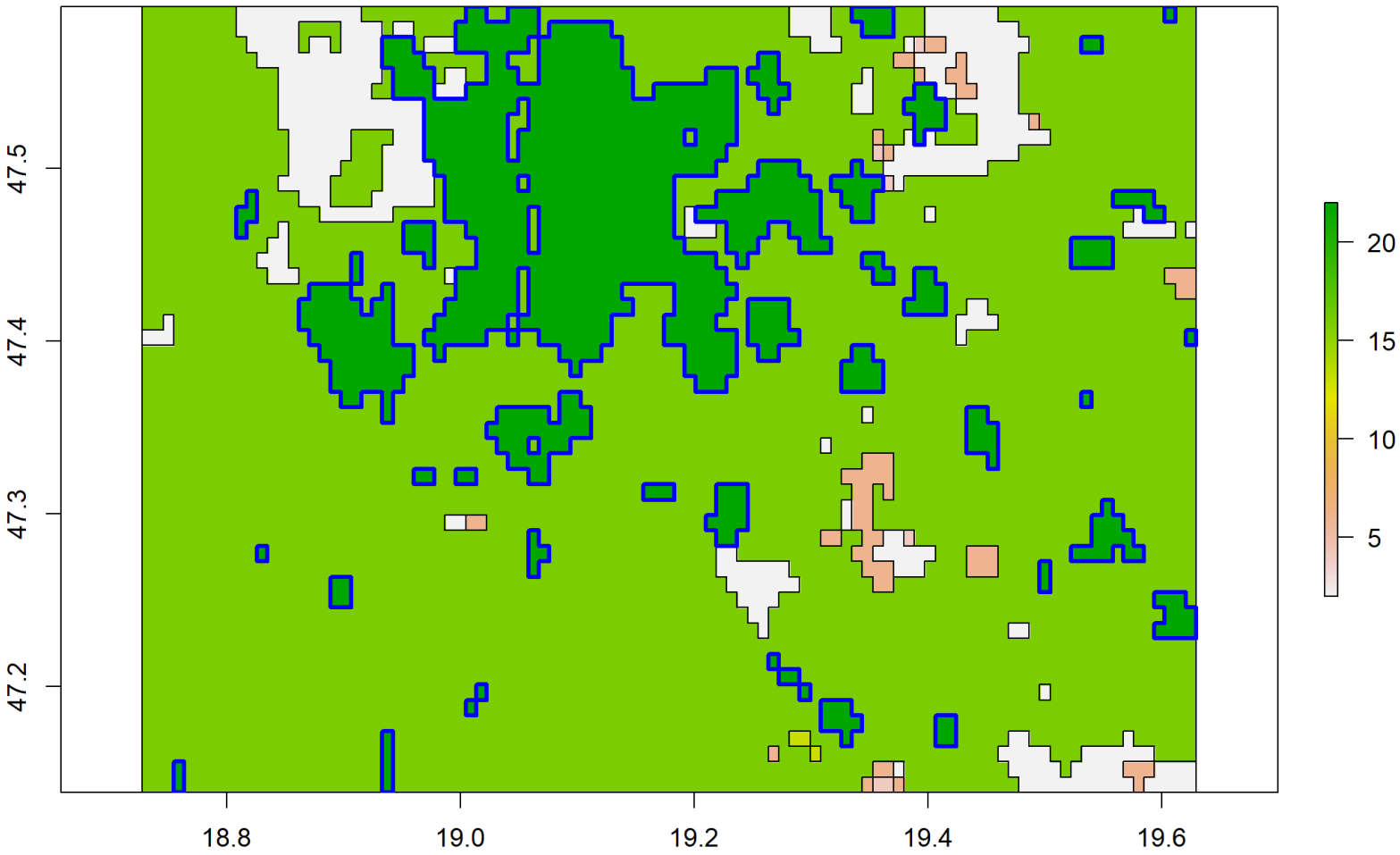
- Hozz létre egy kivágatot a “felszinboritas” nevű raszterből, amely Budapest környékét tartalmazza (200–250. sorok, 350–450. oszlopok), és jelenítsd meg.
- Készíts a raszterből `POLYGON` typusú Simple Features-t úgy, hogy az azonos kategóriába eső blokkokat vond össze.
- Az eredményként kapott poligonok geometriáját add hozzá az ábrához.
- Add még hozzá kék, háromszoros vastagságú körvonallal azon poligonok geometriáját, amelyek az új, “layer” nevű oszlopban a 22-es értéket (=település) tartalmazzák.



# 9. feladat (házi) – megoldás

```
felszinboritas_kivagat <- felszinboritas[200:250, 350:450,
 drop = FALSE]
plot(felszinboritas_kivagat)
felszin_poligonkent_osszevonva <-
 st_as_sf(rasterToPolygons(x = felszinboritas_kivagat,
 dissolve = TRUE))
plot(st_geometry(felszin_poligonkent_osszevonva), add =
 TRUE)
plot(st_geometry(felszin_poligonkent_osszevonva[felszin_poligonkent
 == 22, ]), border = "blue", lwd = 3, add = TRUE)
```

# 9. feladat (házi) – megoldás



# Raszter mátrixszá átalakítása

Alapértelmezett eset:

```
matrixkent <- rasterToPoints(x = szamraszter2, spatial =
 FALSE)
class(matrixkent)
```

```
[1] "matrix" "array"
```

# Raszter mátrixszá átalakítása

```
head(matrixkent)
```

```
          x     y layer
[1,] 0.025 0.975     2
[2,] 0.075 0.975     4
[3,] 0.125 0.975     2
[4,] 0.175 0.975     1
[5,] 0.225 0.975     2
[6,] 0.275 0.975     3
```

# Raszter ponttá átalakítása

A `spatial = TRUE` eset:

```
pontokkent <- rasterToPoints(x = szamraszter2, spatial =
 TRUE)
class(pontokkent)
```

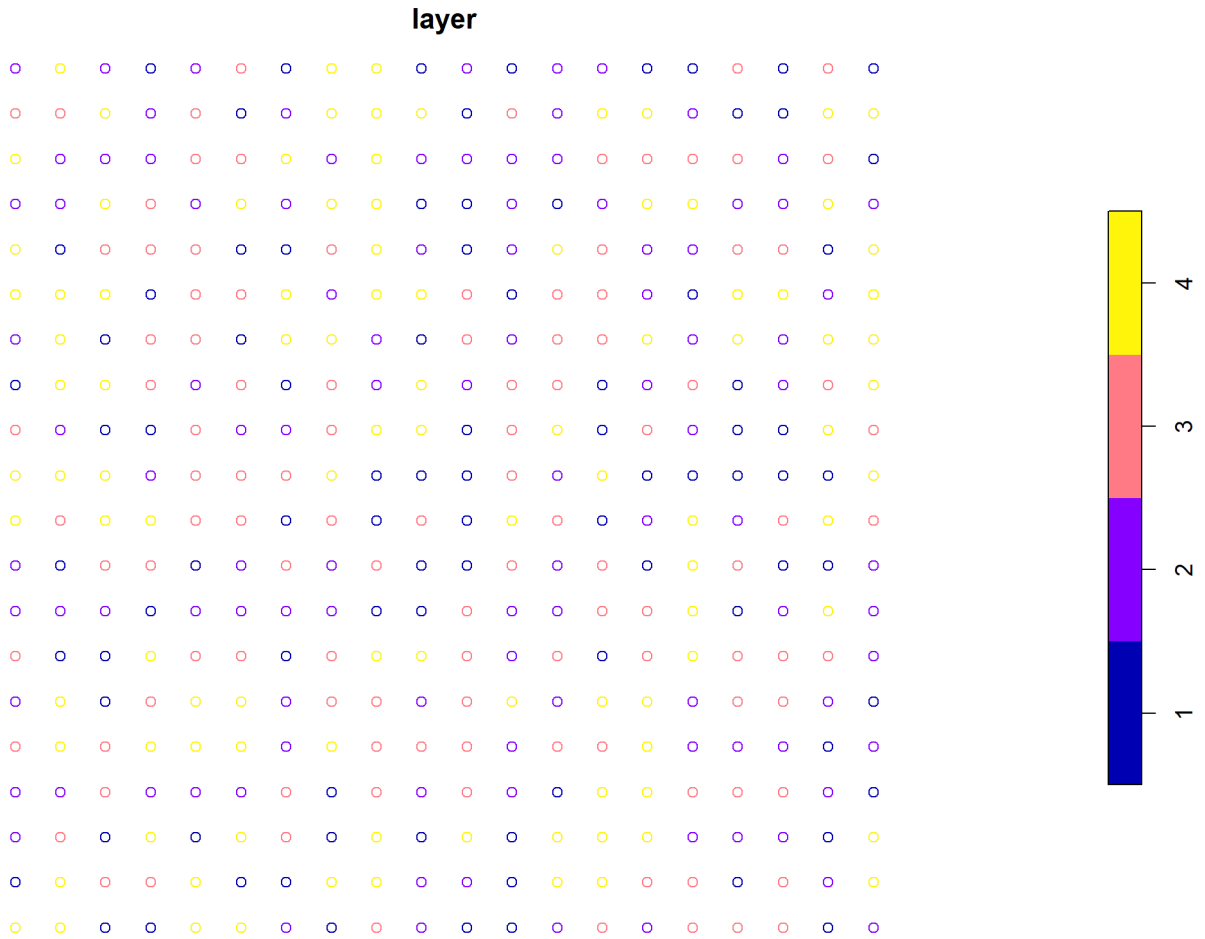
```
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

```
pontokkent <- st_as_sf(pontokkent)
class(pontokkent)
```

```
[1] "sf"         "data.frame"
```

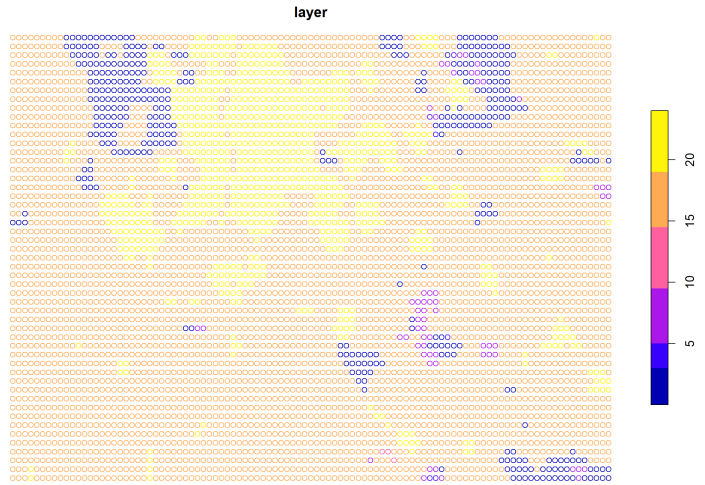
# Raszter ponttá átalakítása

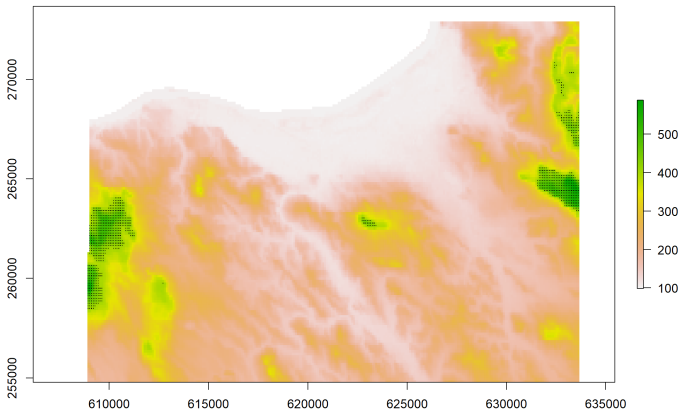
```
plot(pontokkent)
```



# 10. feladat (órai)

- Alakítsd át a korábban kivágott, Budapest környéki felszínborítást `POINT` typusú Simple Features-zé, majd jelenítsd meg.
- Az “esztergom” nevű domborzatmodellt is alakítsd pontokká.
- Jelenítsd meg a magasságot Esztergom környékén, majd add ehhez az ábrához egyszerű pontkarakterrel jelölve azon pontok geometriáját, amelyek 400 méternél magasabban találhatóak.

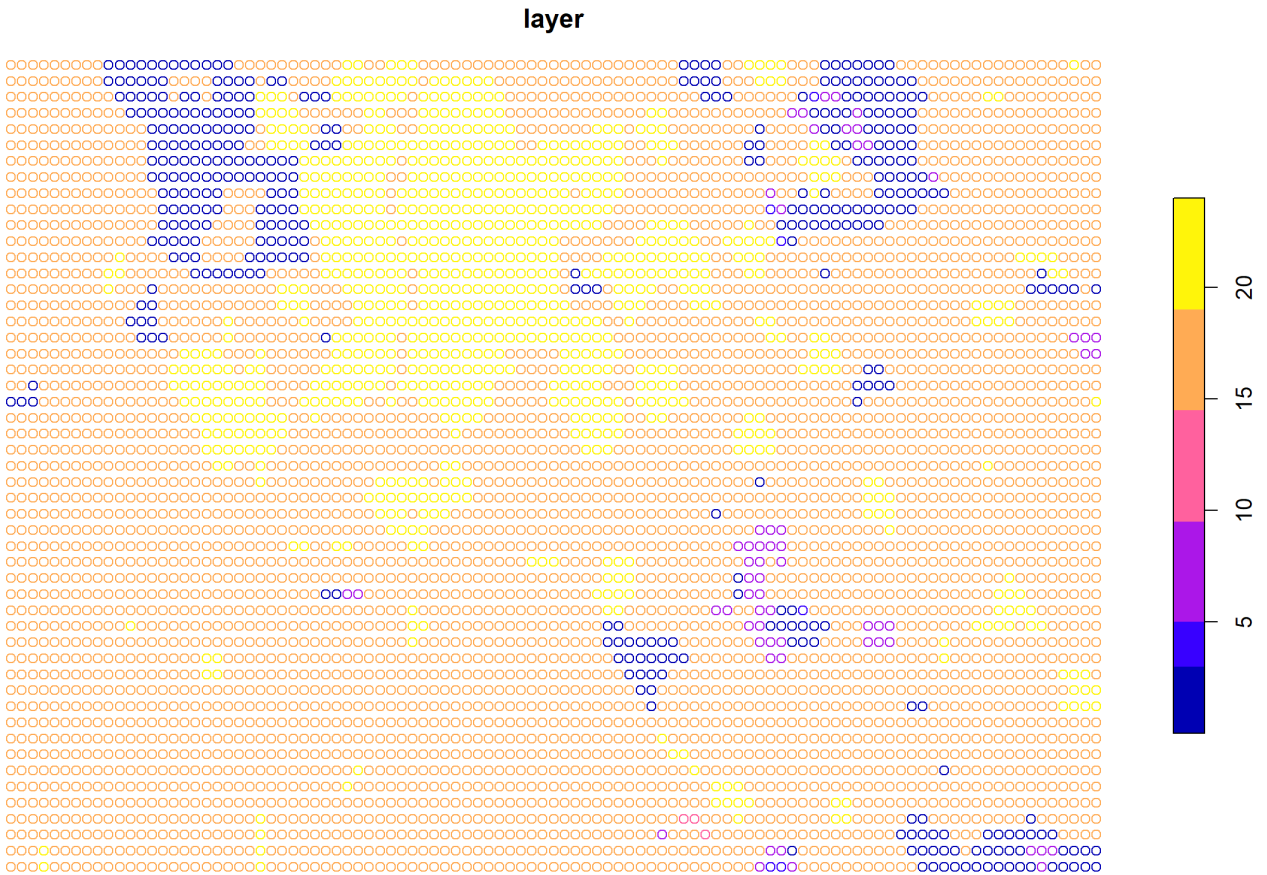


# 10. feladat (órai) – megoldás

```
felszin_pontokkent <- st_as_sf(rasterToPoints(x =
 felszinboritas_kivagat, spatial = TRUE))
```

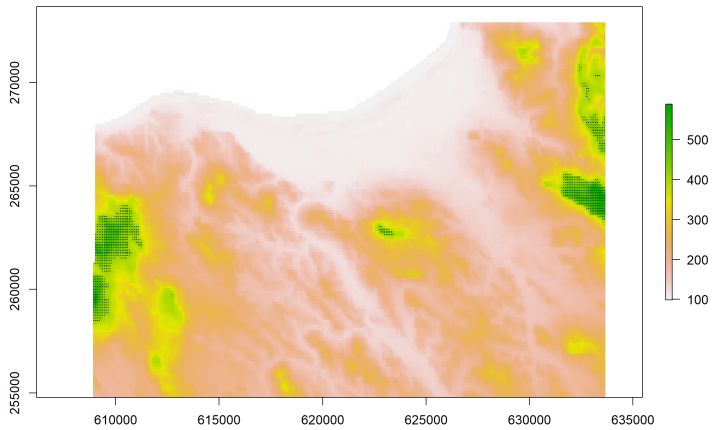
```
plot(felszin_pontokkent)
```



# 10. feladat (órai) – megoldás

```
esztergom_pontokkent <- st_as_sf(rasterToPoints(x =
 esztergom, spatial = TRUE))
```

```
plot(esztergom)
plot(st_geometry(esztergom_pontokkent[
 esztergom_pontokkent$layer > 400, ]), pch = ".", add =
 TRUE)
```




(A pontok olyan sűrűn helyezkednek el, hogy - megjelenítési beállításoktól függően - akár folytonos fekete felületnek is tűnhetnek.)

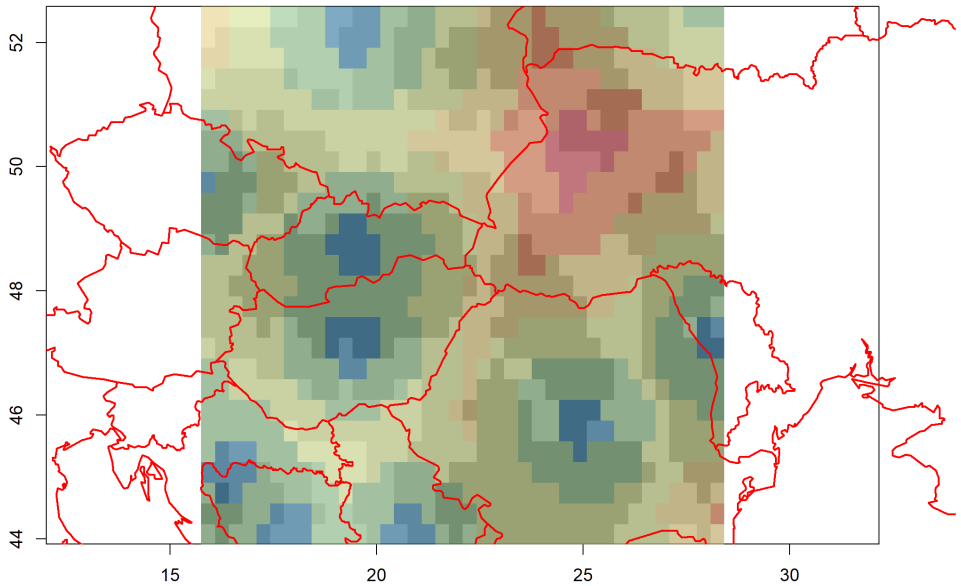
# 11. (összefoglaló) feladat (házi)

- Vágd ki a “csapadek” nevű raszter 450-550. soraiba és 2350-2500. oszlopaiba tartozó területet.
- Csökkents ennek a kivágatnak a felbontását mindkét irányban negyedére, a 4×4-es blokkok legnagyobb értékét továbbörökítve.
- Mekkora most a vízszintes és függőleges felbontása?
- Készíts egy olyan rasztert, amely minden cellájában azt mutatja, hogy az milyen távol esik az interpolált raszter déli (135-225°) irányba csökkenő értékű celláitól.
- Jelenítsd meg a távolságrasztert evvel a szürke színskála 5 színét alkalmazva, jelmagyarázat nélkül. Az ismeretlen értékek fekete színt kapjanak.
- Olvasd be a világ országait tartalmazó “orszagok_osszes.RData” fájlt.
- Képezz egy újabb rasztert, amely azt mutatja, hogy az egyes cellái milyen távol esnek az országokból (MULTIPOLYGON-okból) képzett POLYGON-ok geometriai középpontjaitól.

...

# 11. (összefoglaló) feladat (házi)

- Készíts egy 6 színből álló színskálát a `RColorBrewer` csomag "Spectral" nevű palettájából.
- Az előbbi képvászonhoz új, 0,5-ös átlátszatlansággal, az új színskálát alkalmazva add hozzá ezt a rasztert jelmagyarázat nélkül.
- jelenítsd meg fölöttük piros, kétszeres vastagságú körvonallal az országokat.



# 11. (összefoglaló) feladat (házi) – megoldás

```
csapadek_reszlet <- csapadek[450:550, 2350:2500, drop =
 FALSE]
csapadek_reszlet <- aggregate(x = csapadek_reszlet, fact =
 4, fun = max)
```

```
res(csapadek_reszlet)
```

```
[1] 0.3333333 0.3333333
```

```
del_fele_csokken <- terrain(x = csapadek_reszlet, opt =
 "aspect", unit = "degrees")
del_fele_csokken[del_fele_csokken[] < 135 |
 del_fele_csokken[] > 225] <- NA
tavolsag_ismert_cellatol <- distance(x = del_fele_csokken)
```

# 11. (összefoglaló) feladat (házi) – megoldás

```
plot(tavolsag_ismert_cellatol, colNA = "black", col =
 gray.colors(5), legend = FALSE)
load("orszagok_osszes.RData")
tavolsagok_orszagkozeppontoktol <-
 distanceFromPoints(object = csapadek_reszlet, xy =
 st_centroid(st_cast(orszagok_osszes, "POLYGON")))
library(RColorBrewer)
szinskala <- rev(brewer.pal(n = 6, name = "Spectral"))
plot(tavolsagok_orszagkozeppontoktol, col = szinskala,
 alpha = 0.5, legend = FALSE, add = TRUE)
plot(st_geometry(orszagok_osszes), border = "red", lwd =
 2, add = TRUE)
```

# 11. (összefoglaló) feladat (házi) – megoldás