Example calculations

To reproduce the results on your own computer, install and attach the following packages:

```
library(raceland)
library(raster)
library(sf)
library(tmap)
library(dplyr)
```

You also need to download and extract the `data.zip` file containing the example data.

```
temp_data_file = tempfile(fileext = ".zip")
download.file("https://github.com/Nowosad/raceland-bp1/raw/master/data.zip",
              destfile = temp_data_file,
              mode = "wb")
unzip(temp_data_file)
```

Input data

The presented approach requires a set of rasters, where each raster represents one of five race-groups: Asians, Blacks, Hispanic, others, and Whites. In this example, we use data limited to the city of Cincinnati, Ohio.

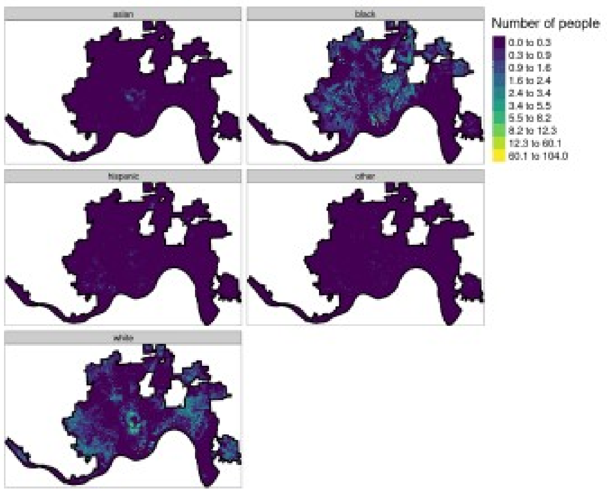
```
list_raster = dir("data", pattern = ".tif$", full.names = TRUE)
race_raster = stack(list_raster)
```

We also use vector data containing the city borders to ease the understanding of the results.

```
cincinnati = read_sf("data/cincinnati.gpkg")
```

We can visualize the data using the **tmap** package:

```
tm_race = tm_shape(race_raster) +
    tm_raster(style = "fisher",
              n = 10,
              palette = "viridis",
              title = "Number of people") +
    tm_facets(nrow = 3) +
    tm_shape(cincinnati) +
    tm_borders(lwd = 3, col = "black")
tm_race
```



The above maps show the distribution of people from different race-groups in Cincinnati. Each, 30 by 30 meters, cell represents a number of people living in this area. Data was obtained from http://sil.uc.edu/cms/index.php?id=socscape-data and preprocessed using the instructions at https://cran.r-project.org/web/packages/raceland/vignettes/raceland-intro3.html.

Basic example

Our goal is to measure racial diversity and racial segregation for different places in the city. We can use the `quanfity_raceland()` function for this purpose.

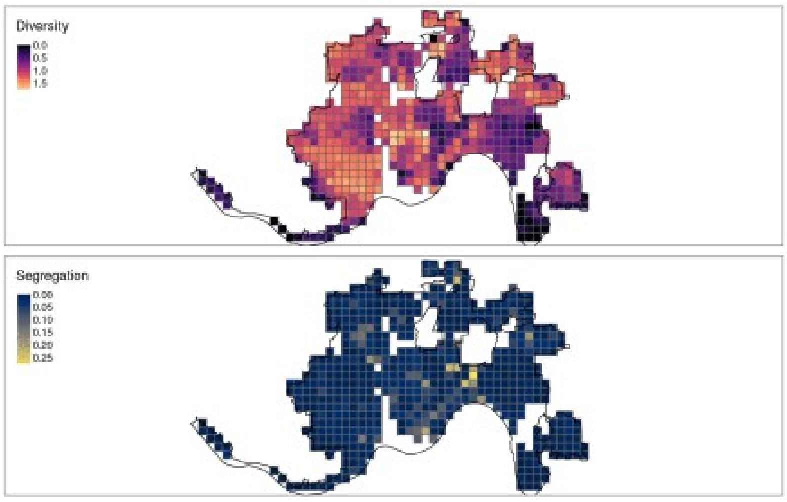
```
results_metrics = quanfity_raceland(race_raster,
                                    n = 30,
                                    window_size = 10,
                                    fun = "mean",
                                    size = 20,
                                    threshold = 0.75)
head(results_metrics)
## Simple feature collection with 6 features and 4 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 978285 ymin: 1858035 xmax: 984885 ymax: 1859235
## CRS:            +proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0
+y_0=0 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
##    row col       ent     mutinf                       geometry
## 30   1  30 1.1050557 0.01559040 POLYGON ((981885 1859235, 9...
## 31   1  31 1.3120756 0.03010253 POLYGON ((982485 1859235, 9...
## 33   1  33 1.1301688 0.01910744 POLYGON ((983685 1859235, 9...
## 34   1  34 1.6320160 0.06155428 POLYGON ((984285 1859235, 9...
## 74   2  24 0.9527805 0.01716798 POLYGON ((978285 1858635, 9...
## 80   2  30 1.4438328 0.04498205 POLYGON ((981885 1858635, 9...
```

It requires several arguments:

- `x` – RasterStack with race-specific population densities assign to each cell
- `n` – a number of realizations
- `window_size` – expressed in the numbers of cells, is a length of the side of a square-shaped block of cells for which local densities will be calculated
- `fun` – function to calculate values from adjacent cells to contribute to exposure matrix, `"mean"` – calculate average values of local population densities from adjacent cells, `"geometric_mean"` – calculate geometric mean values of local population densities from adjacent cells, or `"focal"` assign value from the focal cell
- `size` – expressed in the numbers of cells, is a length of the side of a square-shaped block of cells. It defines the extent of a local pattern
- `threshold` – the share of NA cells to allow metrics calculation

The result is a spatial vector object containing areas of the size of 20 by 20 cells from input data (600 by 600 meters in this example). Its attribute table has five columns – `row` and `col` allowing for identification of each square polygon, `ent` – entropy measuring racial diversity, `mutinf` – mutual information, which is associated with measuring racial segregation, and `geometry` containing spatial geometries.

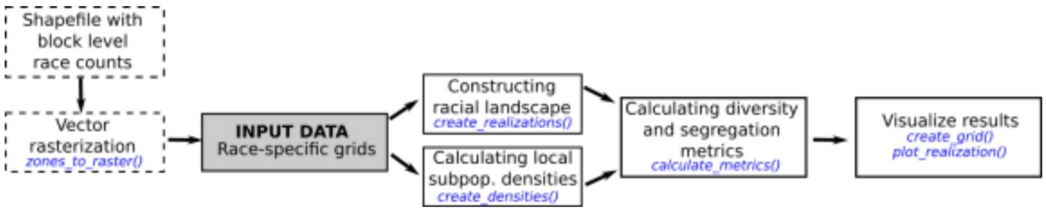
```
diversity_map = tm_shape(results_metrics) +
    tm_polygons(col = "ent",
                title = "Diversity",
                style = "cont",
                palette = "magma") +
    tm_shape(cincinnati) +
    tm_borders(lwd = 1, col = "black")
segregation_map = tm_shape(results_metrics) +
    tm_polygons(col = "mutinf",
                title = "Segregation",
                style = "cont",
                palette = "cividis") +
    tm_shape(cincinnati) +
    tm_borders(lwd = 1, col = "black")
tmap_arrange(diversity_map, segregation_map)
```



The above result present areas with different levels of racial diversity and segregation. Interestingly, there is a low correlation between these two properties. Some areas inside of the city do not have any value attached – this indicates either they are covered with missing values in more than 75% of their areas or nobody lives there.

Extended example

The `quanfity_raceland()` function is a wrapper around several steps implemented in **raceland**, namely `create_realizations()`, `create_densities()`, `calculate_metrics()`, and `create_grid()`. All of them can be used sequentially, as you can see below.



Additionally, the **raceland** package has `zones_to_raster()` function that prepares input data based on spatial vector data with race counts.

Constructing racial landscapes

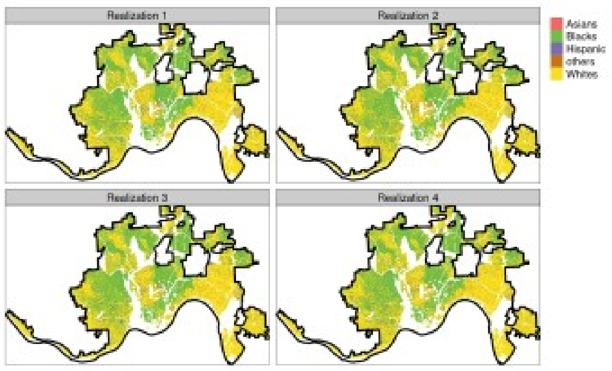
The racial landscape is a high-resolution grid in which each cell contains only inhabitants of a single race. It is constructed using the `create_realizations()` function, which expects a stack of race-specific rasters. Racial composition at each cell is translated into probabilities of drawing a person of a specific race from a cell. For example, if a cell has 100 people, where 90 are classified as Black (90% chance) and 10 as White (10% chance), then we can assign a specific race randomly based on these probabilities.

This approach generates a specified number (`n = 30`, in this case) of realization with slightly different patterns.

```
realizations_raster = create_realizations(race_raster, n = 30)
```

The output of this function is a RasterStack, where each raster contains values from 1 to `k`, where `k` is a number of provided race-specific grids. In this case, we provided five race-specific grids (Asians, Blacks, Hispanic, others, and Whites), therefore the value of `1` in the output object represents Asians, number `2` Blacks, etc.

```
my_pal = c("#F16667", "#6EBE44", "#7E69AF", "#C77213", "#F8DF1D")
tm_realizations = tm_shape(realizations_raster[[1:4]]) +
    tm_raster(style = "cat",
              palette = my_pal,
              labels = c("Asians", "Blacks", "Hispanic", "others", "Whites"),
              title = "") +
    tm_facets(ncol = 2) +
    tm_shape(cincinnati) +
    tm_borders(lwd = 3, col = "black") +
    tm_layout(panel.labels = paste("Realization", 1:30))
tm_realizations
```



The above plot shows four of 30 created realizations and makes it clear that all of them are fairly similar.

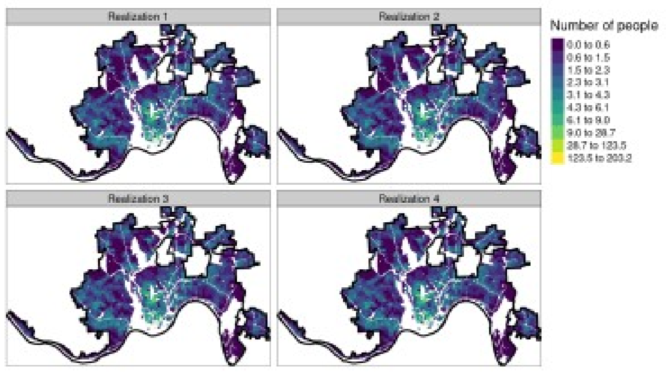
Local densities

Now, for each of the created realization, we can calculate local densities of subpopulations (race-specific local densities) using the `create_densities()` function.

```
dens_raster = create_densities(realizations_raster,
                               race_raster,
                               window_size = 10)
```

The output is a RasterStack with local densities calculated separately for each realization.

```
tm_density = tm_shape(dens_raster[[1:4]]) +
    tm_raster(style = "fisher",
              n = 10,
              palette = "viridis",
              title = "Number of people") +
    tm_facets(ncol = 2) +
    tm_shape(cincinnati) +
    tm_borders(lwd = 3, col = "black") +
    tm_layout(panel.labels = paste("Realization", 1:30))
tm_density
```



Total diversity and segregation

We can use both, realizations and density rasters, to calculate racial diversity and segregation using `calculate_metrics()` function. It calculates four information theory-derived metrics: entropy (`ent`), joint entropy (`joinent`), conditional entropy (`condent`), and mutual information (`mutinf`). As we mentioned before, `ent` is measuring racial diversity, while `mutinf` is associated with racial segregation. These metrics can be calculated for a given spatial scale. For example, setting `size` to `NULL`, as in the example below, calculates the metrics for the whole area of each realization.

```
metr_df = calculate_metrics(x = realizations_raster,
                            w = dens_raster,
                            fun = "mean",
                            size = NULL,
                            threshold = 1)
head(metr_df)
##   realization row col      ent  joinent  condent    mutinf
## 1           1   1   1 1.400229 2.625657 1.225428 0.1748010
## 2           2   1   1 1.398806 2.624101 1.225295 0.1735102
## 3           3   1   1 1.398361 2.623339 1.224978 0.1733824
## 4           4   1   1 1.400530 2.625777 1.225247 0.1752829
## 5           5   1   1 1.395641 2.617376 1.221734 0.1739072
## 6           6   1   1 1.397392 2.616627 1.219235 0.1781572
```

Now, we can calculate average metrics across all realization, which should give more accurate results.

```
metr_df %>%
  summarise(
    mean_ent = mean(ent, na.rm = TRUE),
    mean_mutinf = mean(mutinf)
  )
##   mean_ent mean_mutinf
## 1 1.397863   0.1742165
```

These values could be compared with values obtained by other US cities to evaluate, which cities have high average racial diversity (larger values of `mean_ent`) and which have high average racial segregation (larger values of `mean_mutinf`).

Local diversity and segregation

The information theory-derived metrics can be also calculated for smaller, local scales using the `size` argument. It describes the size of a local area for metrics calculations. For example, `size = 20` indicates that each local area will consist of 20 by 20 cells of the original raster.

```
metr_df_20 = calculate_metrics(x = realizations_raster,
                               w = dens_raster,
                               fun = "mean",
                               size = 20,
                               threshold = 0.75)
```

Now, we can summarize the results for each local area independently (`group_by(row, col)`).

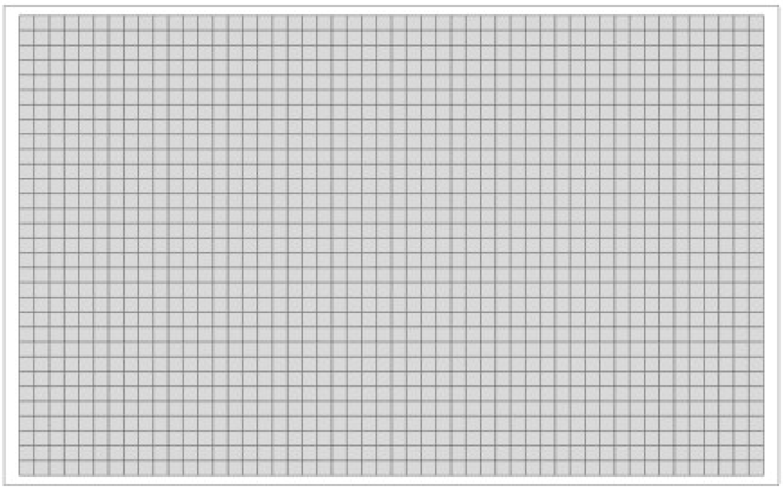
```
smr = metr_df_20 %>%
  group_by(row, col) %>%
  summarize(
    ent_mean = mean(ent, na.rm = TRUE),
    mutinf_mean = mean(mutinf, na.rm = TRUE),
  ) %>%
  na.omit()
head(smr)
## # A tibble: 6 x 4
## # Groups:   row [2]
##      row   col ent_mean mutinf_mean
##
## 1      1    30    1.09        0.0152
## 2      1    31    1.30        0.0356
## 3      1    33    1.12        0.0159
## 4      1    34    1.62        0.0576
## 5      2    24    0.959       0.0195
## 6      2    30    1.44        0.0445
```

Each row in the obtained results relates to some spatial locations. We can create an empty grid with appropriate dimensions using the `create_grid()` function. Its `size` argument expects the same value as used in the `calculate_metrics()` function.

```
grid_sf = create_grid(realizations_raster, size = 20)
```

The result is a spatial vector object with three columns: `row` and `col` allowing for identification of each square polygon, and `geometry` containing spatial geometries.

```
tm_shape(grid_sf) +
    tm_polygons()
```

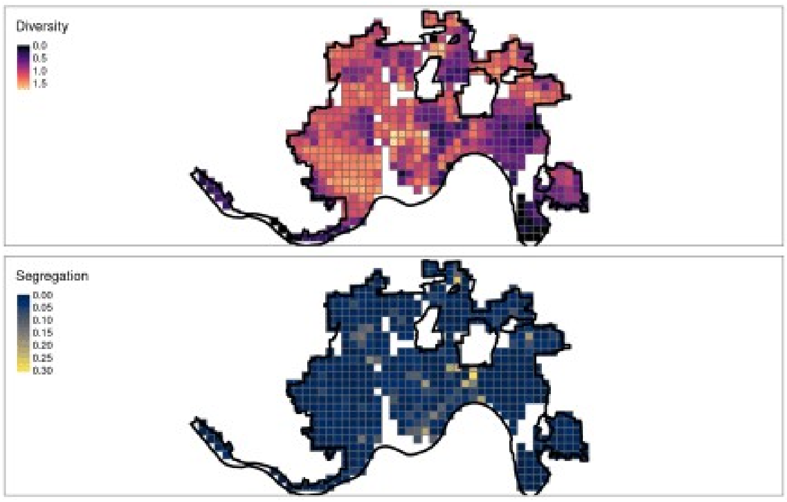


The first two columns,`row` and `col`, can be used to connect the grid with summary results.

```
grid_attr = dplyr::left_join(grid_sf, smr, by = c("row", "col"))
grid_attr = na.omit(grid_attr)
```

Finally, we are able to create two maps. The first one represents racial diversity (larger the value, larger the diversity; the `ent_mean` variable) and the second one shows racial segregation (larger the value, larger the segregation; the `ent_mean` variable).

```
diversity_map = tm_shape(grid_attr) +
    tm_polygons(col = "ent_mean",
                title = "Diversity",
                style = "cont",
                palette = "magma") +
    tm_shape(cincinnati) +
    tm_borders(lwd = 3, col = "black")
segregation_map = tm_shape(grid_attr) +
    tm_polygons(col = "mutinf_mean",
                title = "Segregation",
                style = "cont",
                palette = "cividis") +
    tm_shape(cincinnati) +
    tm_borders(lwd = 3, col = "black")
tmap_arrange(diversity_map, segregation_map)
```



Bonus: visualizing racial landscapes

While the realizations created few steps before represents race spatial distribution fairly well, they do not take the spatial variability of the population densities into consideration. Additional function `plot_realization()` displays a selected realization taking into account not only race spatial distribution, but also the population density.

```
plot_realization(x = realizations_raster[[2]],
                 y = race_raster,
                 hex = my_pal)
```

In its result, darker areas have larger populations, and brighter represent areas less-inhabited areas.

Summary

The **raceland** package implements a computational framework for a pattern-based, zoneless analysis and visualization of (ethno)racial topography. The most comprehensive description of the method can be found in the Racial landscapes – a pattern-based, zoneless method for analysis and visualization of racial topography article published in Applied Geography. Its preprint is available at https://osf.io/preprints/socarxiv/mejz5. Additionally, **raceland** has three extensive vignettes:

- raceland: R package for a pattern-based, zoneless method for analysis and visualization of racial topography – introducing the package and its functions
- raceland: Describing local racial patterns of racial landscapes at different spatial scales – showing how the calculations can be performed at different spatial scales
- raceland: Describing local pattern of the racial landscape using SocScape grids – presenting how to use the **raceland** methods with SocScape race-specific grids to perform analysis for different spatial scales, using the Cook county as an example.