The R programming language has, over the past two decades, evolved substantial spatial data analysis capabilities, and is now one of the most powerful environments for undertaking geographic research using a reproducible command line interface. Currently, dedicated R packages allow to read spatial data and apply a plethora of different kinds of spatial methods in a reproducible fashion.

There are two main[1] spatial data models – spatial vector data and spatial raster data. Natively R does not support spatial data and does not have a definition of spatial classes. Therefore, there had been a need to create R tools able to represent spatial vector and raster data. Spatial classes are slightly different from regular R objects, such as data frames or matrices, as they need to not only store values, but also information about spatial locations and their coordinate reference systems.

Nowadays, the most prominent packages to represent spatial vector data are **sf** (Pebesma 2021a) and its predecessor **sp** (Pebesma and Bivand 2021), however, the **terra** (Hijmans 2021b) package also has its own spatial class for vector data. Spatial raster data can be stored as objects from **terra** (Hijmans 2021b) and its predecessor **raster** (Hijmans 2021a), or alternatively the **stars** package (Pebesma 2021b).

As you could see in our Why R? webinar talk, the spatial capabilities of R constantly expand, but also evolve. New packages are being developed, while old ones are modified or superseded. In this process, new methods are created, higher performance code is added, and possible workflows are expanded. Alternative approaches allow for a (hopefully) healthy competition, resulting in better packages. Of course, having more than one package (with its own spatial class/es) for a vector or raster data model could be problematic, especially for new or inexperienced users.

First, it takes time to understand how different spatial classes are organized. To illustrate this, let's read the same spatial data, `srtm.tif` from the **spDataLarge** package (Nowosad and Lovelace 2021), using **raster** and **stars**. The **raster** object:

```
raster_file_path = system.file("raster/srtm.tif", package =
"spDataLarge")
library(raster)
srtm_raster = raster(raster_file_path)
srtm_raster
## class      : RasterLayer
## dimensions : 457, 465, 212505  (nrow, ncol, ncell)
## resolution : 0.0008333333, 0.0008333333  (x, y)
## extent     : -113.2396, -112.8521, 37.13208, 37.51292  (xmin, xmax,
ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : srtm.tif
## names      : srtm
## values     : 1024, 2892  (min, max)
```

The **stars** object:

```
library(stars)
srtm_stars = read_stars(raster_file_path)
srtm_stars
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##            Min. 1st Qu. Median     Mean 3rd Qu. Max.
## srtm.tif  1024    1535   1837 1842.548    2114 2892
## dimension(s):
##   from  to  offset        delta refsys point values x/y
## x    1 465 -113.24  0.000833333 WGS 84 FALSE   NULL [x]
## y    1 457 37.5129 -0.000833333 WGS 84 FALSE   NULL [y]
```

Secondly, other packages with methods we want to use, could only accept one specific spatial class, but not the other. For example, the current version of the **sabre** package (Nowosad and Stepinski 2019) (0.3.2) accepts objects from the **raster** package, but not ones from **terra** or **stars**[2]. The `partitions1` and `partitions2` objects are of the `RasterLayer` class from **raster**, so the `vmeasure_calc()` function works correctly.

```
library(sabre)
library(raster)
data("partitions1")
data("partitions2")
vmeasure_calc(partitions1, partitions2)
## The SABRE results:
##
##  V-measure: 0.36
##  Homogeneity: 0.32
##  Completeness: 0.42
##
##  The spatial objects can be retrieved with:
##  $map1 - the first map
##  $map2 - the second map
```

However, when the input object (representing the same spatial data!) is of the `SpatRaster` class from **terra**, the calculation results in error.

```
vmeasure_calc(partitions1_terra, partitions1_terra)
## Error in UseMethod("vmeasure_calc") :
##  no applicable method for 'vmeasure_calc' applied to an object of
class "SpatRaster"
```

Some packages, such as **tmap** (Tennekes 2021), accept many R spatial classes, however, this takes a lot of effort from package creators to make it possible and to maintain it. Gladly, a number of functions exist that allow to convert between different R spatial classes. Using them, we can work with our favorite spatial data representation, switch to some other representation just for a certain calculation, and then convert the result back into our class. The next two sections showcase how to move between different spatial vector and raster data classes in R.

Spatial vector data

The `world.gpkg` file from the **spData** (Bivand, Nowosad, and Lovelace 2020) contains spatial vector data with world countries.

```
world_path = system.file("shapes/world.gpkg", package = "spData")
```

Now, we can read this file, for example, as an `sf` object, and convert it into other spatial vector data classes.

```
library(sf)
library(sp)
library(terra)

# read as sf
world = read_sf(world_path)

# sf to sp
world_sp1 = as(world, "Spatial")

# sf to terra vect
world_terra1 = vect(world)

# sp to terra vect
world_terra2 = vect(world_sp1)

# sp to sf
world_sf2 = st_as_sf(world_sp1)

# terra vect to sf
world_sf3 = st_as_sf(world_terra1)

# terra vect to sp
world_sp2 = as(world_terra1, "Spatial")
```

In summary, `st_as_sf()` converts other classes into `sf`, `vect()` transform other classes into **terra**'s `SpatVector`, and with `as(x, "Spatial")` it is possible to get **sp**'s vectors.

FROM/TO	**sf**	**sp**	**terra**
sf		*as(x, “Spatial”)*	*vect()*
sp	*st_as_sf()*		*vect()*
terra	*st_as_sf()*	*as(x, “Spatial”)*	

Spatial raster data

The `srtm.tif` file from the **spDataLarge** (Nowosad and Lovelace 2021) contains a raster elevation model for the Zion National Park in the USA.

```
srtm_path = system.file("raster/srtm.tif", package = "spDataLarge")
```

Now, we can read this file, for example, as a **raster** object, and convert it into other spatial vector data classes.

```
library(raster)
library(stars)
library(terra)

srtm_raster1 = raster(srtm_path)

# raster to terra
srtm_terra1 = rast(srtm_raster1)
```

```
# terra to raster
srtm_raster2 = raster(srtm_terra1)

# raster to stars
srtm_stars1 = st_as_stars(srtm_raster1)

# stars to raster
srtm_raster2 = as(srtm_stars1, "Raster")

# terra to stars
srtm_stars2 = st_as_stars(srtm_terra1)

# stars to terra?
srtm_raster2a = as(srtm_stars1, "Raster")
srtm_terra1a = rast(srtm_raster2a)
```

As you can see – in most cases, we can just use one function to move from one class to another. The only exception is from **stars** to **terra** – currently, there is not a dedicated custom function for that conversion. I plan to update this post, if/when this function will be created.

FROM/TO	**raster**	**terra**	**stars**
raster		*rast()*	*st_as_stars()*
terra	*raster()*		*st_as_stars()*
stars	*raster()*	*as(x, “Raster”) + rast()*	

Summary

This blog post summarizes how to move between different R spatial vector and raster classes. All of the functions mentioned above have one role: to change classes of input objects. They do not, however, change geometries or underlining values in the data.