Packages, Version Control, & Testing

2023-10-03

R projects

RStudio is a useful interface for R programming that facilitates the creation and execution of R scripts and RMarkdown files. RStudio allows the creation of projects, which are essentially directories that contain associated files. One common file structure convention for R projects include a `R/` folder for R scripts, a `data/` folder with raw and processed subdirectories for all data necessary, a `doc/` folder for documentation, and an `output/` folder for outputs. Additionally, R projects often include a `README.md` document describing the project. More complex projects may include a `src/` folder for C/C++ code and/or a a `tests/` folder for testing scripts.

Version control

Version control is a crucial part of the software development project, crucial for reproducibility and collaboration (including collaboration with one's future self!). Git + GitHub is a popular version control system that allows version control both locally and online in public or private repositories. I have created private GitHub repositories for both Statistical Computing 1 and Statistical Methods 1 to store my code for these courses. An example of a public repository of mine is the repository for my R package `hull2spatial` that I discuss in the remainder of this portfolio.

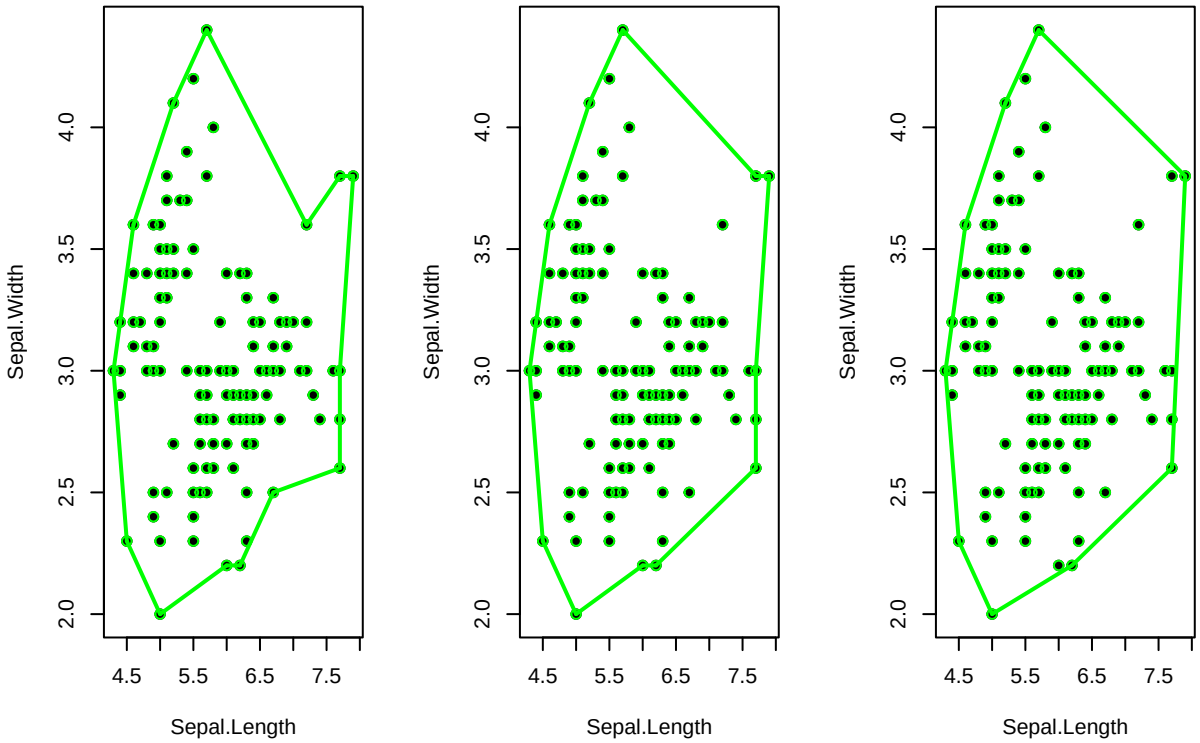
Creating an R package

For this portion of the portfolio, I have been working to improve an R package I have been working on for the last few years called `hull2spatial`. The package can be found in this GitHub repository: https://github.com/babichmorrowc/hull2spatial. The purpose of the package is to convert objects produced by the `alphahull` R package into objects that are compatible with the `sp` package.

The `alphahull` package creates 2 types of shapes around a set of points: α-shapes and α-hulls. Both types of shapes depend on the parameter α: when α is high, the resulting shape is closer to a convex hull, while when α is low, the resulting shape is more concave. α-shapes are composed of straight lines, while α-hulls can include arcs. We can see an example of an α-shape here:

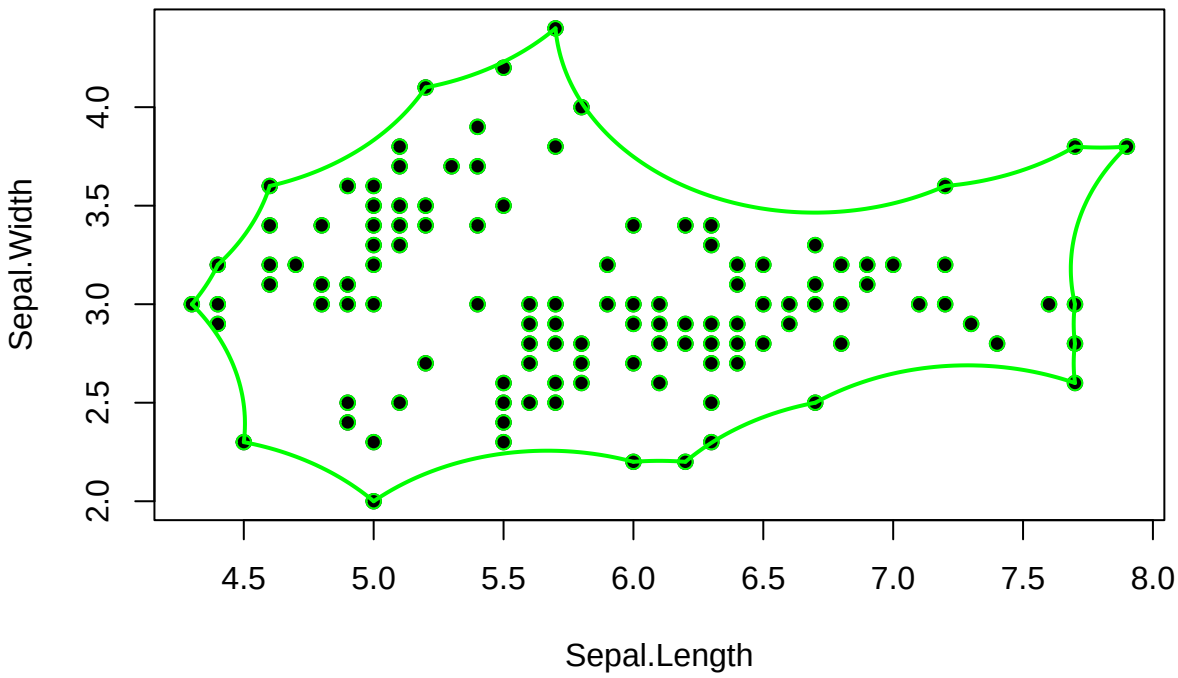
```
library(alphahull)
data(iris)
library(dplyr)

# Create a dataframe of sepal length and width
sepal_data <- iris %>%
  filter(!is.na(Sepal.Length) & !is.na(Sepal.Width)) %>%
  select(Sepal.Length, Sepal.Width) %>%
  distinct() # remove duplicates

# Create 3 alpha shapes varying alpha
ashape_1 <- ashape(sepal_data, alpha = 1)
```

```
ashape_2 <- ashape(sepal_data, alpha = 2)
ashape_4 <- ashape(sepal_data, alpha = 4)
# Create a 3-paneled figure
par(mfrow = c(1,3))
# Plot
plot(sepal_data, pch = 19)
plot(ashape_1, col = "green", add = TRUE)
plot(sepal_data, pch = 19)
plot(ashape_2, col = "green", add = TRUE)
plot(sepal_data, pch = 19)
plot(ashape_4, col = "green", add = TRUE)
```



α-hulls, on the other hand, include arcs:

```
ahull_1 <- ahull(sepal_data, alpha = 1)
plot(sepal_data, pch = 19)
plot(ahull_1, col = "green", add = TRUE)
```



These objects are of special classes in R that do not interface well with other packages for spatial data manipulation, such as `sf`:

```
class(ashape_1)
```

```
## [1] "ashape"
```

```
class(ahull_1)
```

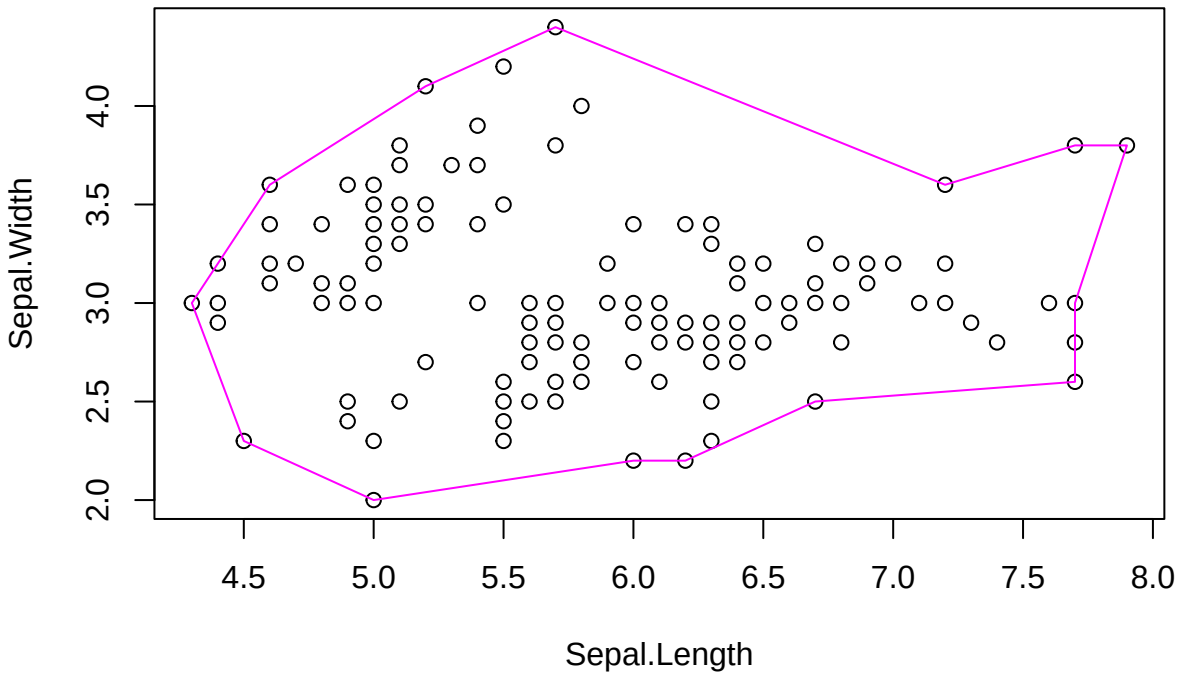
```
## [1] "ahull"
```

My package `hull2spatial` rectifies this issue by providing several functions that convert these objects into spatial objects:

```
library(devtools)
install_github("babichmorrowc/hull2spatial")
library(hull2spatial)
library(sp)

# Convert ashape_1 to SpatialPolygons object
polygon_ashape_1 <- ashape2poly(ashape_1)
class(polygon_ashape_1)
```

```
## [1] "SpatialPolygons"
## attr(,"package")
## [1] "sp"
```

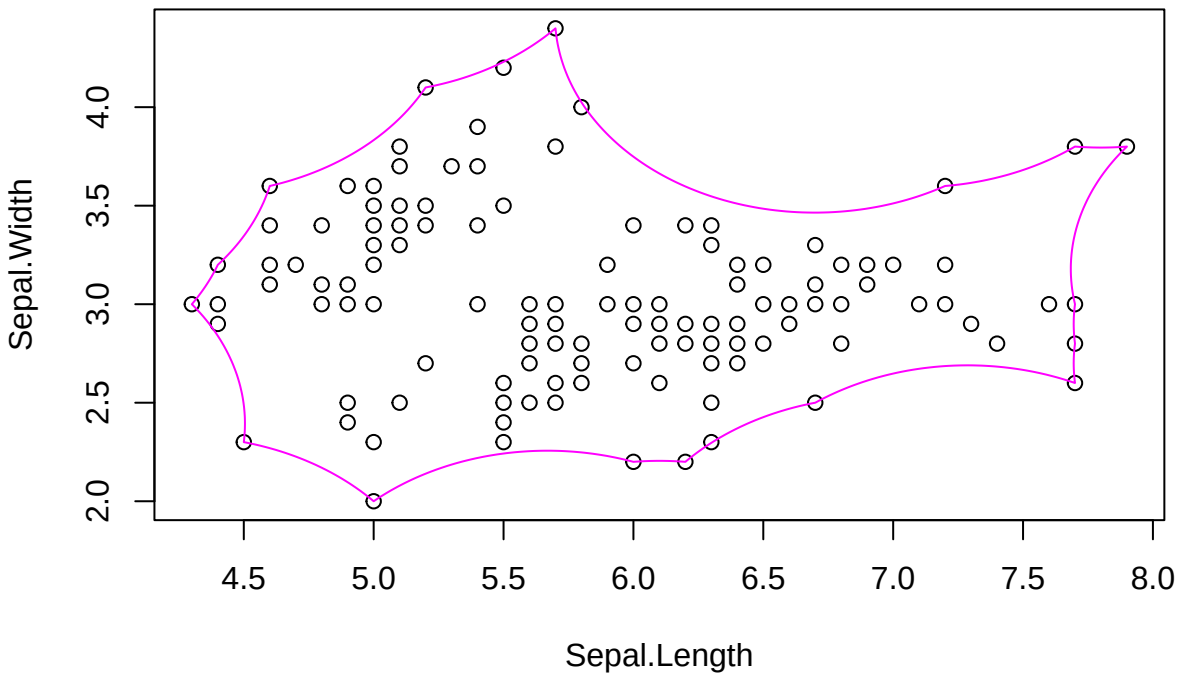
```
plot(sepal_data)
plot(polygon_ashape_1, border = "magenta", add = TRUE)
```




```
# Convert ahull_1 to SpatialPolygons object
polygon_ahull_1 <- ahull2poly(ahull_1)
class(polygon_ahull_1)
```

```
## [1] "SpatialPolygons"
## attr(,"package")
## [1] "sp"
```

```
plot(sepal_data)
plot(polygon_ahull_1, border = "magenta", add = TRUE)
```



See this blog post for more detail on the motivation behind this package.

Package improvements

Following our lecture about packages in R, I made the following improvements to my `hull2spatial` package to add a license and testing to help improve the robustness and reproducibility of my code.

Adding a license

I hadn't previously added a license to `hull2spatial`, so I used the `usethis` package to add one. After doing some reading about license selection here, I selected the GPLv3license, which is a copyleft license that stipulates that everything using my code must remain open source. The `usethis` package has a built-in function `use_gpl_license()` that will automatically set up the appropriate licensing when run within the project containing the package.

Writing tests

I also wanted to add some tests to my package using the `testthat` package. The `usethis` package automatically creates testing files in the proper format. By running `usethis::use_test()` when the relevant file of functions is open, it automatically created the test file using the naming convention `test-<r-file-name>.R` and put it inside the `tests/testthat` folder. In the case of my package, the test files can all be found here. By going to Build > Test Package within RStudio, I can run all of the tests I have written and find out how many tests have passed. This is equivalent to running `devtools::test()` in the console.

I also set up GitHub Actions testing by running `usethis::use_github_action_check_standard()`, as mentioned in the course website here. I did receive the following warning message:

```
Warning message:
`use_github_action_check_standard()` was deprecated in usethis 2.2.0.
  Please use the `check-standard` argument of `use_github_action()` instead.
```

which indicates that as of the most recent version of the `usethis` package, it is best to run `use_github_action("check-standard")` instead. I made a small pull request to update this in the course website.

I noticed that the GitHub Actions workflows were failing, so I worked on a series of fixes, mostly having to do with the need to explicitly note the package from which a given function was taken, e.g. `alphahull::ahull`. I also found that by running `devtools::check_rhub()` in the RStudio Console, I could run CRAN checks on the package before pushing my changes (which is faster than waiting for the entire GitHub Actions checks to finish after pushing). After several rounds of this process, I was able to end up with passing R-CMD-check (see here).