GIS Plotting in R

Announcements

- This week’s lab is short to give time to absorb and finish up labs 1-3.
- If you haven’t submitted your final project pre-proposal, please do this asap!

Why learn to plot in R as opposed to other software?

- ArcPro is EXPENSIVE
- QGIS has its own learning curve.
- R plots:
  - Accept complex logic that is hard to achieve and reproduce easily in a GUI
  - Self documenting methods
  - Auditable & Reproducible
  - Standardized look and feel
  - Highly customizable

Base R Plotting

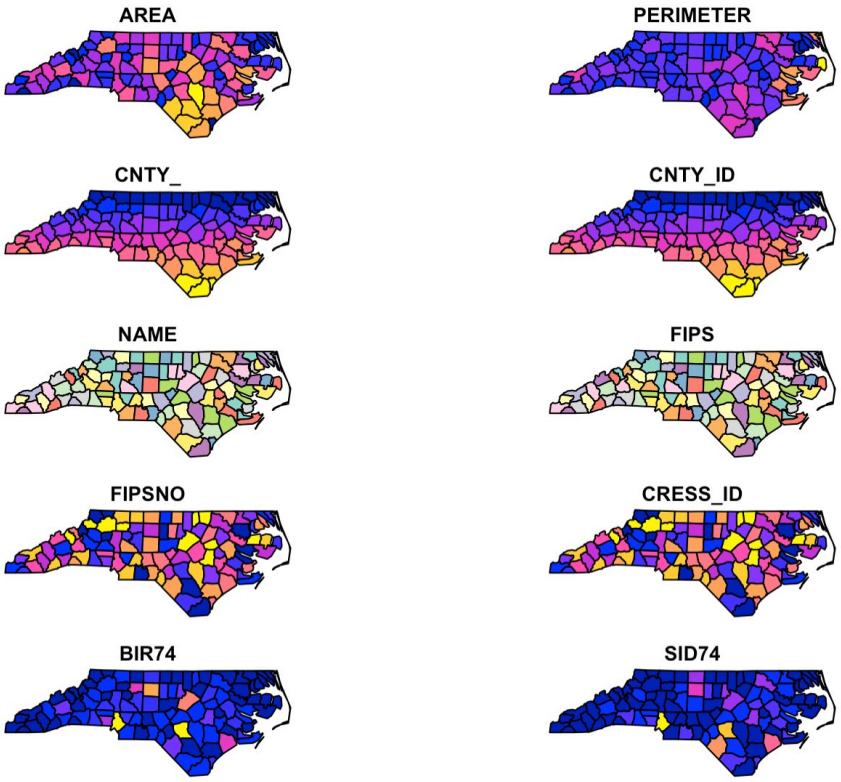
- plot(sf)
- plot(raster)
- Pros
  - Very quick and easy syntax
  - Little effort to take a look at your data
- Cons
  - Looks quite bad in my opinion
  - Customization leaves much to be desired

Useful, but not pretty

- > plot(nc[5])

Even worse if I don’t specify an attribute

- > plot(nc)



1. ggplot2

ggplot2

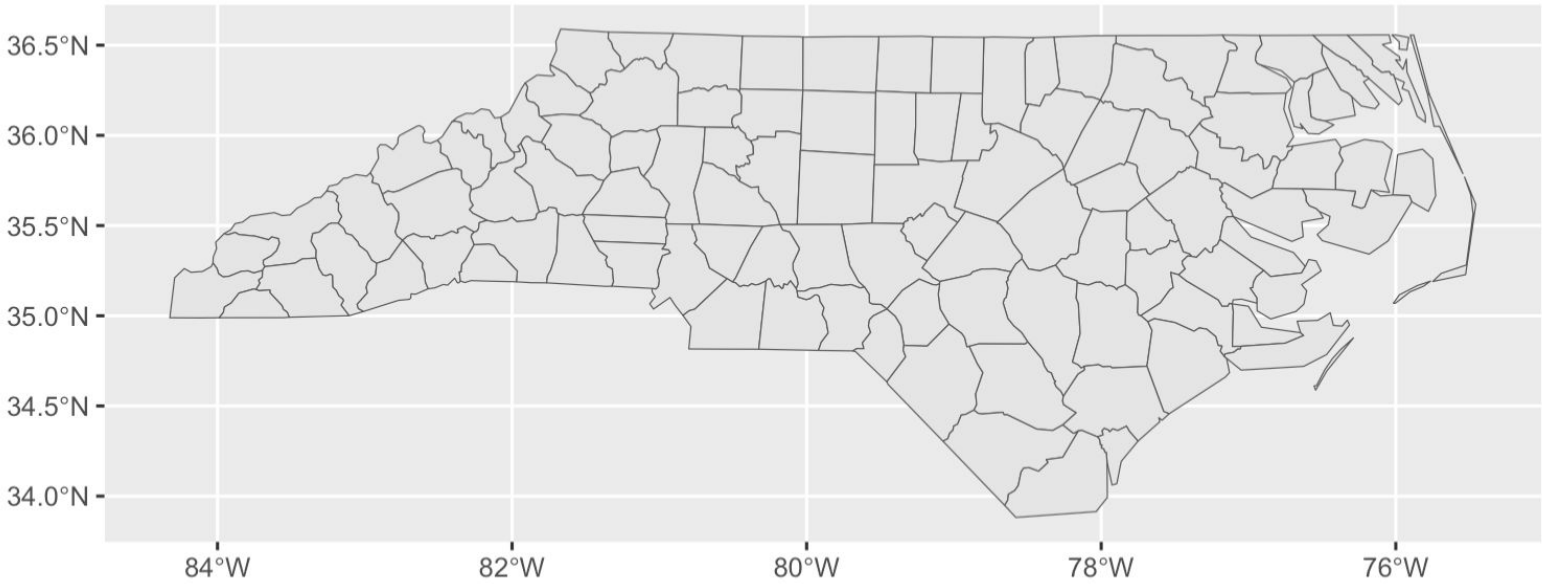
- “Grammar of Graphics”
- Pros
  - Highly customizable plots
  - Integrated with sf and terra
- Cons
  - Learning curve
  - Less specialized for maps compared to tmap

ggplot2 - Mapping Essentials Cheat Sheet

1. **ggplot()** Initiator Function: The foundation of the plot, specifying the default dataset and aesthetic mappings.
2. **geom_sf()** Geom Layers: Define the type of plot or the geometric objects
   - Points, lines, polygons
3. **aes()** Aesthetic Mappings: Specify how variables in the data are mapped to visual properties (like color, size, shape).
4. **labs()** Labels: Add titles, subtitles, captions, and axis labels.
5. **theme()** Theme Function: Modify non-data ink on the plot, including backgrounds, grid lines, and text elements.

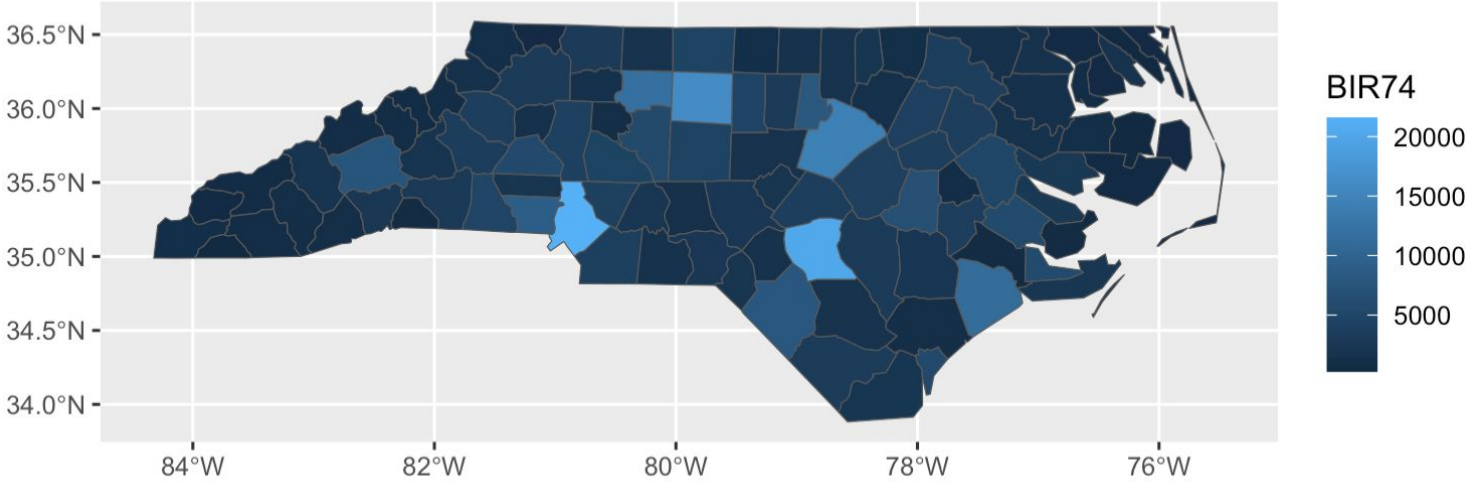
ggplot2 Customization - Basic Plot

```
ggplot(data = nc) +
  geom_sf()
```



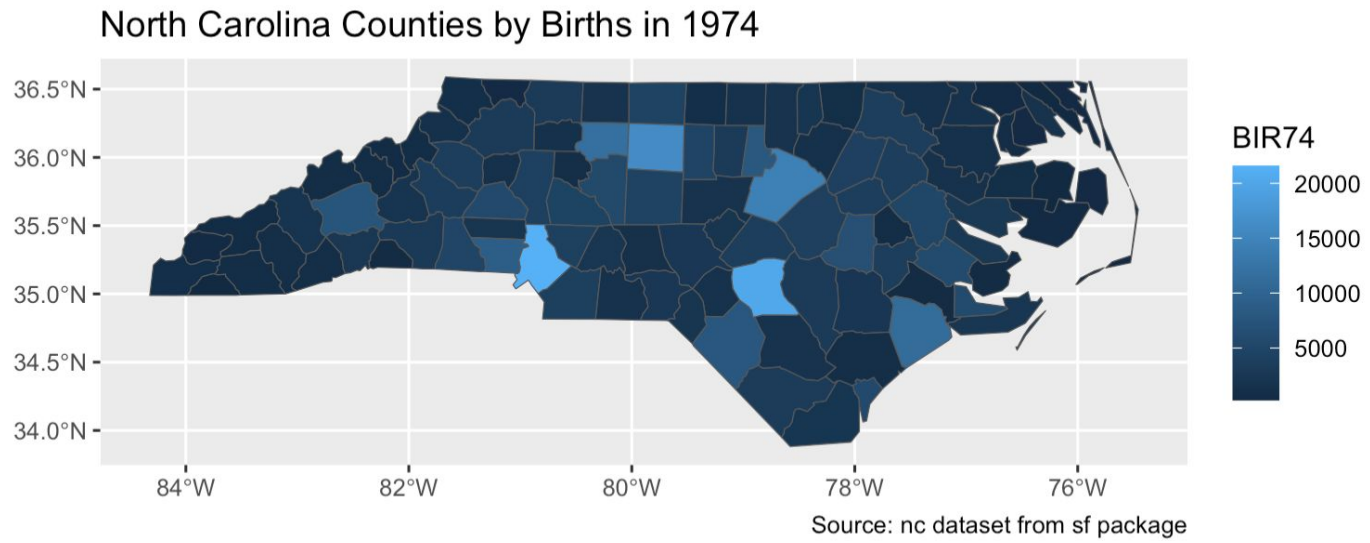
ggplot2 Customization - Fill Aesthetics

```
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74))
```



ggplot2 Customization - Labels

```
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74)) +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package")
```

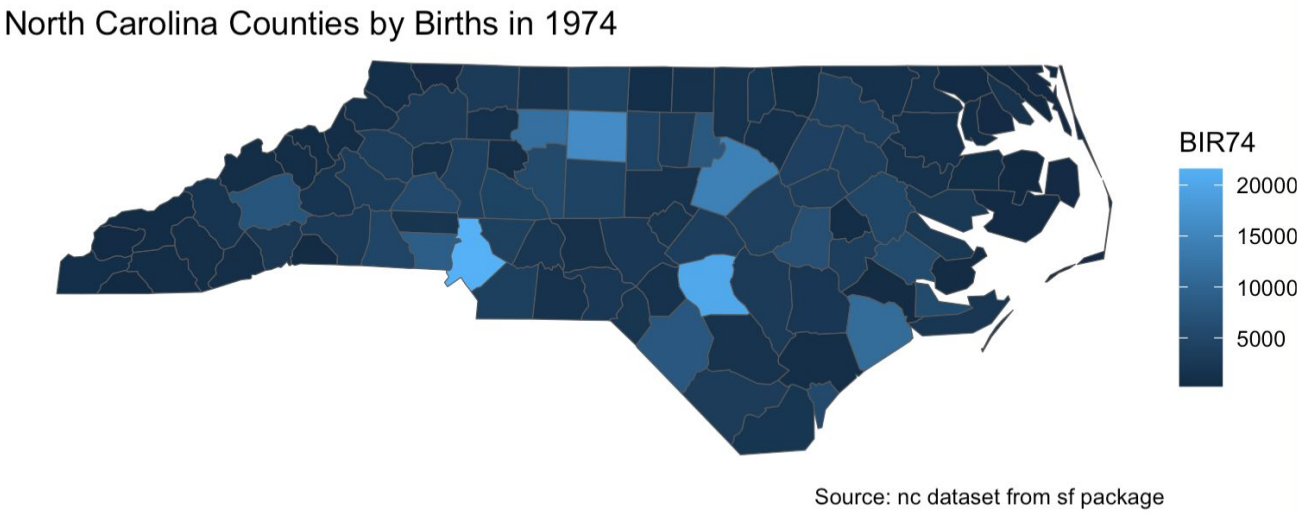


ggplot2 Customization - Preset Themes Cheat Sheet

- theme_grey()
- theme_bw()
- theme_linedraw()
- theme_light()
- theme_dark()
- theme_void()
- theme_minimal()
- theme_classic()

ggplot2 Customization - Preset Themes Example

```
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74)) +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package") +
  theme_void()
```

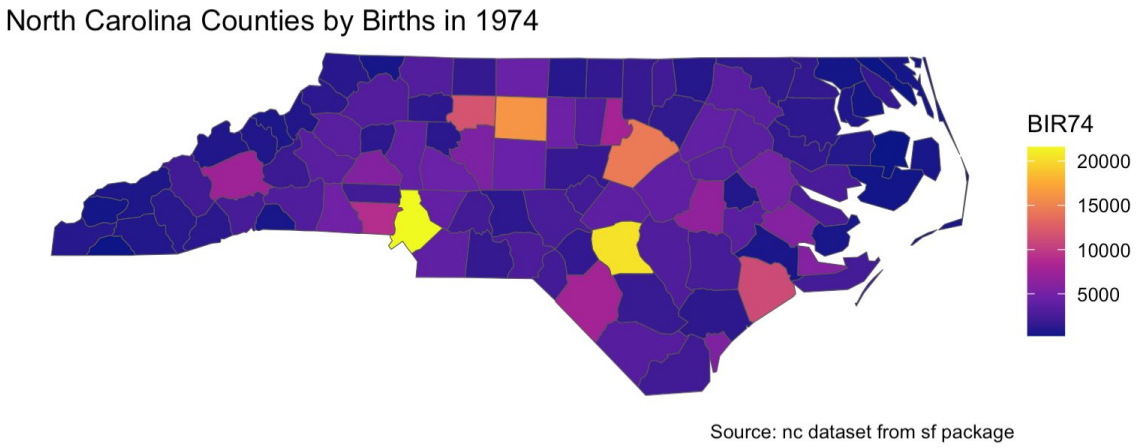


ggplot2 Customization - Changing Choropleth Colors

- Viridis Color Scales
  - scale_fill_viridis_c(): Great for people with color vision deficiencies.
- Gradient Color Scales
  - scale_fill_gradient(): Creates a two-color gradient (specify low and high colors).
  - scale_fill_gradient2(): Creates a three-color gradient, with a midpoint color.
  - scale_fill_gradientn(): Creates a gradient from n colors.
- Scientific Journal Color Scales (via ggsci package)
  - scale_fill_npg(): Nature Publishing Group palette.
  - scale_fill_aaas(): Science (AAAS) palette.

ggplot2 Customization - Changing Choropleth Colors

```
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74)) +
  scale_fill_viridis_c(option = "C") +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package") +
  theme_void()
```



ggplot2 Customization - theme() Input Parameters

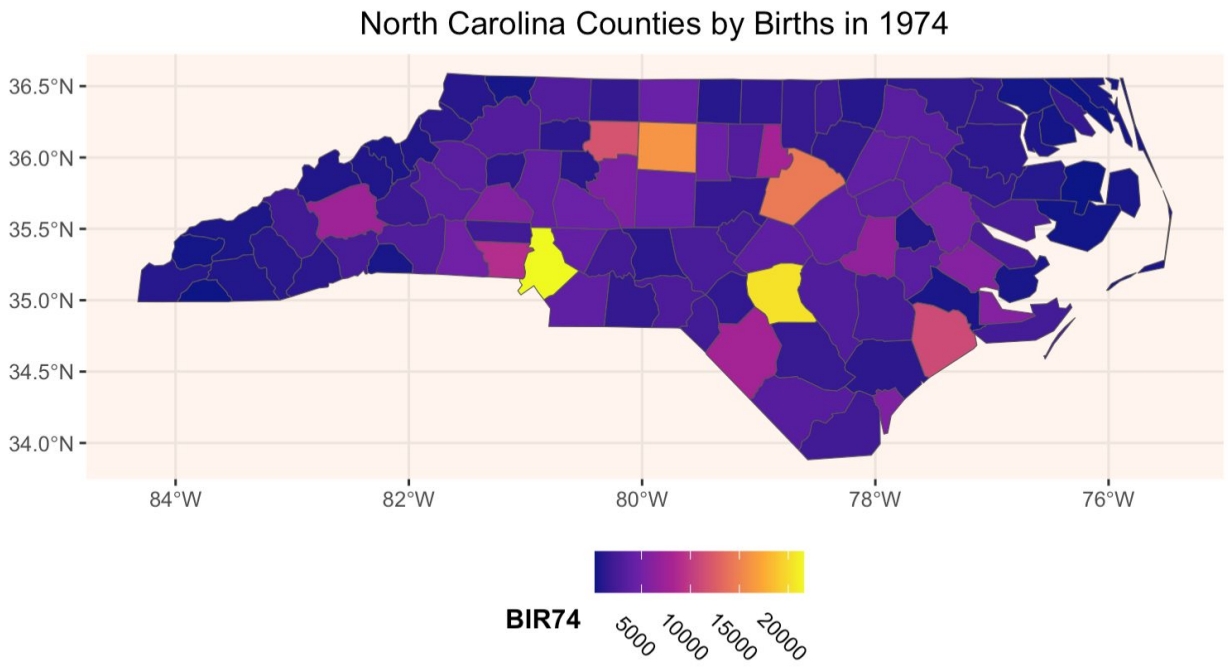
- theme(): allows you to highly customize your plots on a per element basis
- element_text(): text.
- element_line(): lines.
- element_rect(): borders and backgrounds.
- element_blank(): draws nothing, and assigns no space.

General Rule: If the element you're trying to inspect is a line, use element_line() for customization. If the element is text, use element_text(). If you want to erase something, use element_blank().

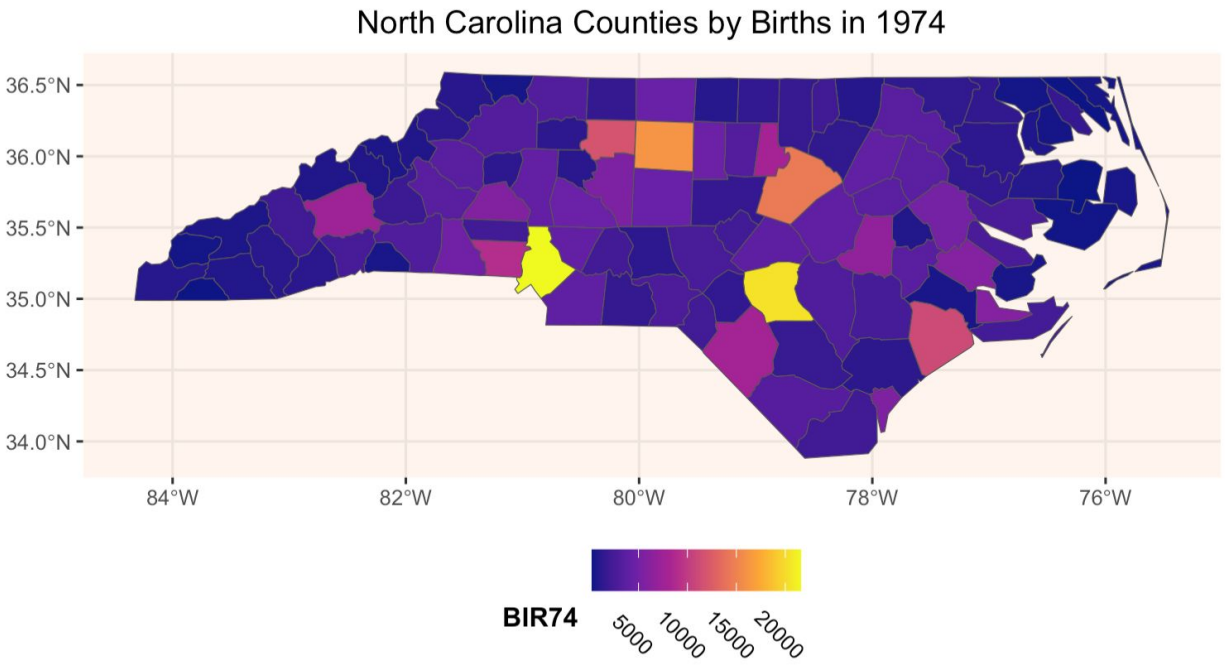
ggplot2 Customization - Custom Themes

```
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74)) +
  scale_fill_viridis_c(option = "C") +
  theme(legend.position = "bottom",
        plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "seashell"),
        panel.grid.major = element_line(color = "seashell2", linewidth = 0.5),
        panel.grid.minor = element_line(color = "seashell2", linewidth = 0.25),
        legend.title     = element_text(face = "bold"),
        legend.text      = element_text(angle = -45,
                                        margin = margin(t = 10, unit = "pt")),
        panel.border = element_blank())+
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package")
```

ggplot2 Customization - Custom Themes



Big difference!



Source: nc dataset from sf package

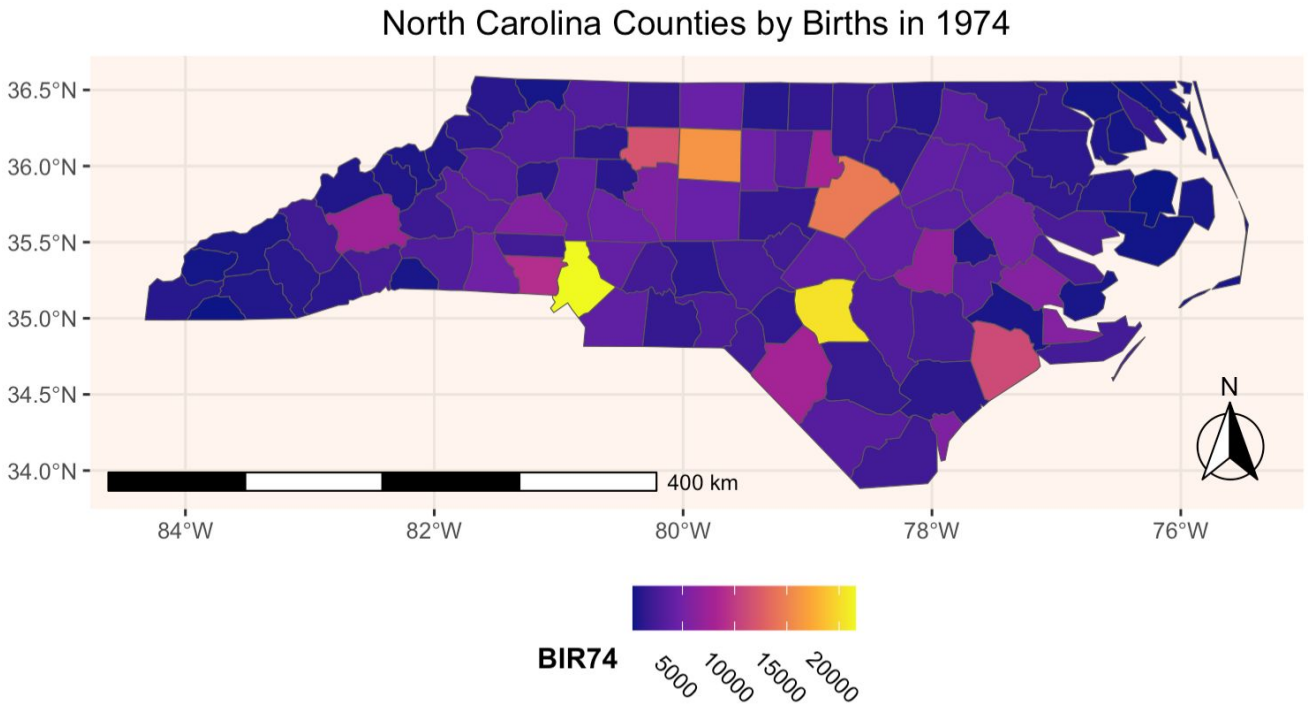
ggplot2 Customization - Compass & Scale Bar

- ggplot2 cannot generate a compass and scale bar out of the box
- The **ggspatial package is required**
  - annotation_scale()
  - annotation_north_arrow()

ggplot2 Customization - Compass & Scale Bar Example

```r
require(ggspatial)
ggplot(data = nc) +
  geom_sf(aes(fill = BIR74)) +
  scale_fill_viridis_c(option = "C") +
  theme(legend.position = "bottom",
        plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "seashell"),
        panel.grid.major = element_line(color = "seashell2", linewidth = 0.5),
        panel.grid.minor = element_line(color = "seashell2", linewidth = 0.25),
        legend.title     = element_text(face = "bold"),
        legend.text      = element_text(angle = -45,
                                        margin = margin(t = 10, unit = "pt")),
        panel.border     = element_blank()) +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package") +
  annotation_scale(location = "bl", width_hint = 0.5) + # Add a scale bar
  annotation_north_arrow(location = "br",
                         which_north = "true",
                         pad_x = unit(0.1, "in"),
                         pad_y = unit(0.1, "in"),
                         style = north_arrow_fancy_orienteering) # Add a compass
```

ggplot2 Customization - Compass & Scale Bar Example



Source: nc dataset from sf package

ggplot2 Customization - Basemaps

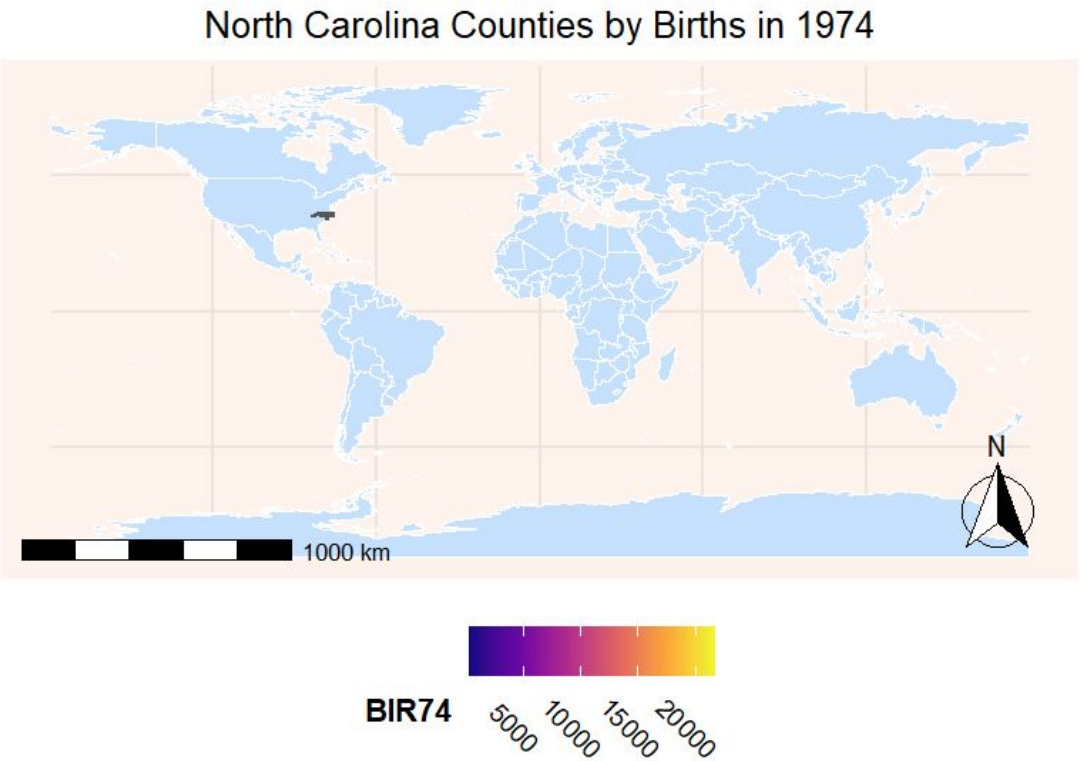
- ggmap() is a spatial extension package to ggplot2
  - Enables basemap functionality
  - Caveat: You typically need an API key, which is not great…
- rnaturalearth + rnaturalearthdata is great for basemaps
  - Less involved basemaps than ggmap(), but no API needed

ggplot2 Customization - Suboptimal Basemap

```
require(rnaturalearth)
require(rnaturalearthdata)
world <- ne_countries(scale = "medium", returnclass = "sf")

ggplot() +
  geom_sf(data = world, fill = "slategray1", color = "white") +
  geom_sf(data = nc, aes(fill = BIR74)) +
  scale_fill_viridis_c(option = "C") +
  theme(legend.position = "bottom",
        plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill  = "seashell"),
        panel.grid.major = element_line(color = "seashell2", linewidth = 0.5),
        panel.grid.minor = element_line(color = "seashell2", linewidth = 0.25),
        legend.title     = element_text(face = "bold"),
        legend.text      = element_text(angle = -45, margin = margin(t = 10, unit = "pt")),
        panel.border     = element_blank()) +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package") +
  annotation_scale(location = "bl", width_hint = 0.5) +
  annotation_north_arrow(location = "br",
                         which_north = "true",
                         pad_x = unit(0.1, "in"),
                         pad_y = unit(0.1, "in"),
                         style = north_arrow_fancy_orienteering)
```

ggplot2 Customization - Suboptimal Basemap



ggplot2 Customization - Better Basemap

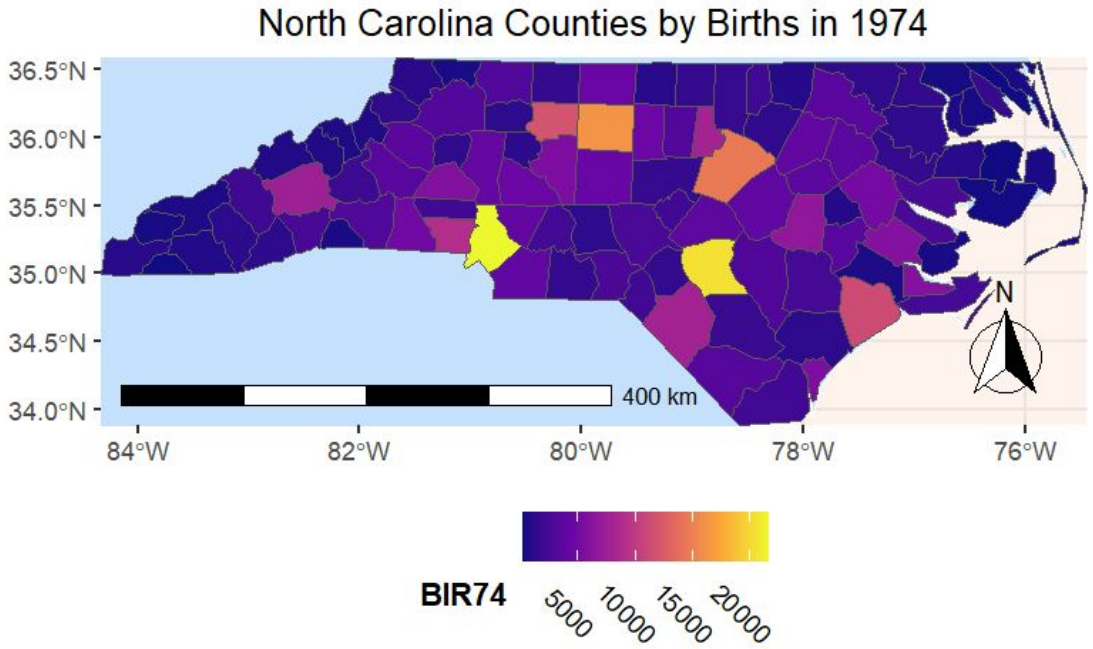
```
nc_bounds <- st_bbox(nc)
xlim <- c(nc_bounds["xmin"], nc_bounds["xmax"])
ylim <- c(nc_bounds["ymin"], nc_bounds["ymax"])
```

← Specify map extent w/ st_bbox()

```
ggplot() +
  geom_sf(data = world, fill = "slategray1", color = "white") +
  geom_sf(data = nc, aes(fill = BIR74)) +
  scale_fill_viridis_c(option = "C") +
  theme(legend.position = "bottom",
        plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill  = "seashell"),
        panel.grid.major = element_line(color = "seashell2", linewidth = 0.5),
        panel.grid.minor = element_line(color = "seashell2", linewidth = 0.25),
        legend.title     = element_text(face = "bold"),
        legend.text      = element_text(angle = -45, margin = margin(t = 10, unit = "pt")),
        panel.border     = element_blank()) +
  labs(title = "North Carolina Counties by Births in 1974",
       caption = "Source: nc dataset from sf package") +
  annotation_scale(location = "bl", width_hint = 0.5) +
  annotation_north_arrow(location = "br",
                         which_north = "true",
                         pad_x = unit(0.1, "in"),
                         pad_y = unit(0.1, "in"),
                         style = north_arrow_fancy_orienteering) +
  coord_sf(xlim = xlim, ylim = ylim, expand = FALSE)  # Focus on NC
```

← Apply map extent w/ coord_sf()

ggplot2 Customization - Better Basemap



Source: nc dataset from sf package

2. tmap

tmap

- Built on the foundations of ggplot2
- Expands functionality of ggplot to specialize in mapping
- Pros:
  - Designed for mapping
  - Contains mapping specific functionality like north arrow and scale bars
  - Highly customizable
- Cons:
  - Only really useful for mapping
  - Learning curve, but much less than ggplot2

tmap - Layer Basics

- Calling Layers
  - **tm_shape()** : Prepares the data layer to be visualized in subsequent tm_* function
  - tm_polygons() : Adds polygons to map
  - tm_lines() : Adds lines to map
  - tm_dots() : Adds points to map
  - tm_raster() : Adds raster to map

tmap - Calling Layers Example

```
> tm_shape(nc)
Error: no layer elements defined after tm_shape
```

```
> tm_polygons(nc)
Error: tm_shape element missing
```

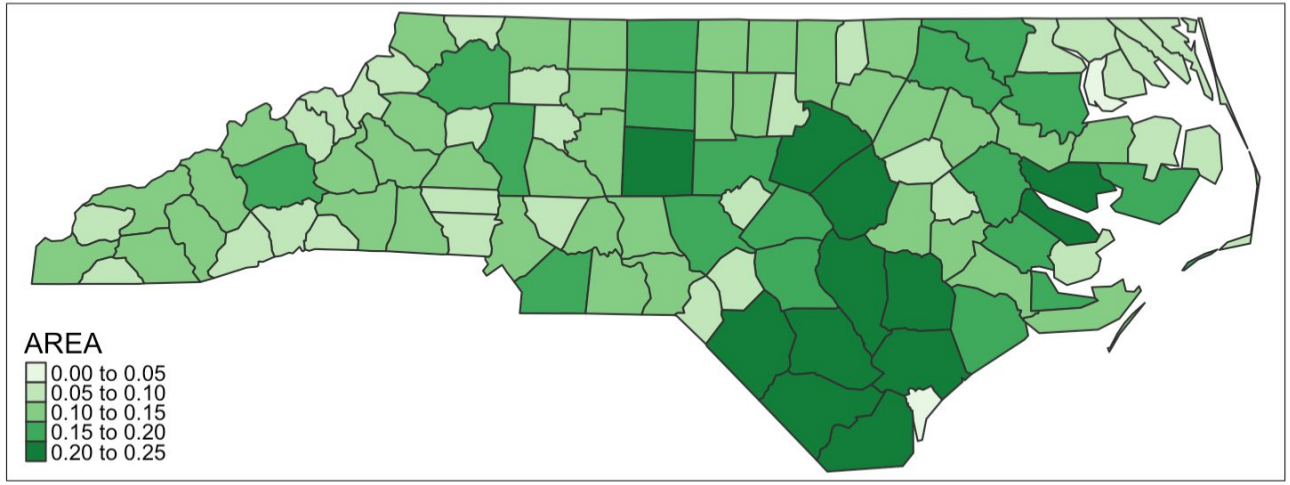
```
> tm_shape(nc) +
+     tm_polygons()
```

tmap - Customization Basics

- Customizing
  - tm_borders() : Adds borders around spatial features without filling them
  - tm_fill() : Fills feature with color, which can be based on an attribute of the data
  - tm_layout() : Customizes the layout and styling of the map
  - tm_compass() : Adds a compass
  - tm_scale_bar() : Adds a scale bar
  - tmap_style() : Sets the overall aesthetics of the map

tmap - Customization Progression 1

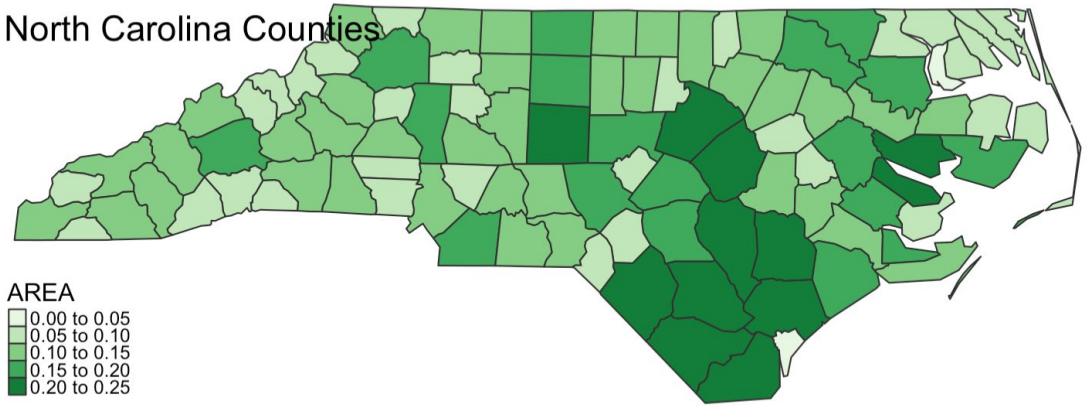
```
tm_shape(nc) +
  tm_borders() +
◆ tm_fill(col = "AREA")
```



tmap - Customization Progression 2

```
tm_shape(nc) +
  tm_borders() +
  tm_fill(col = "AREA") +
  tm_layout(frame = FALSE,
            title = "North Carolina Counties")
```

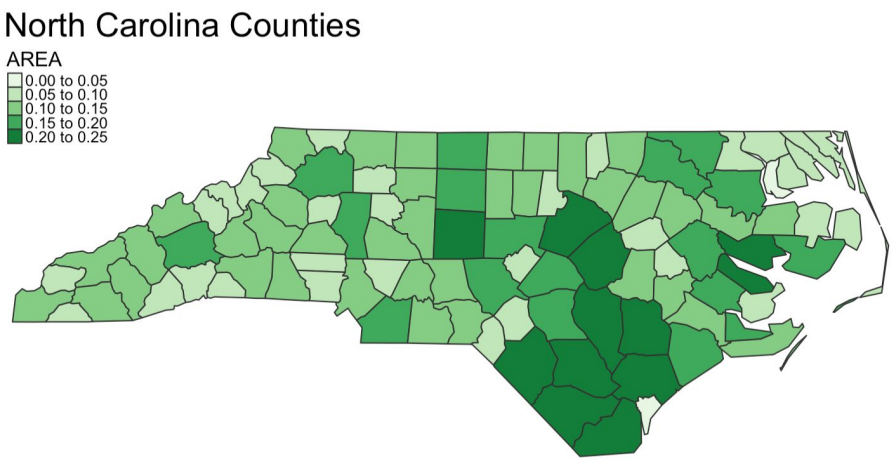
ew…



tmap - Customization Progression 3

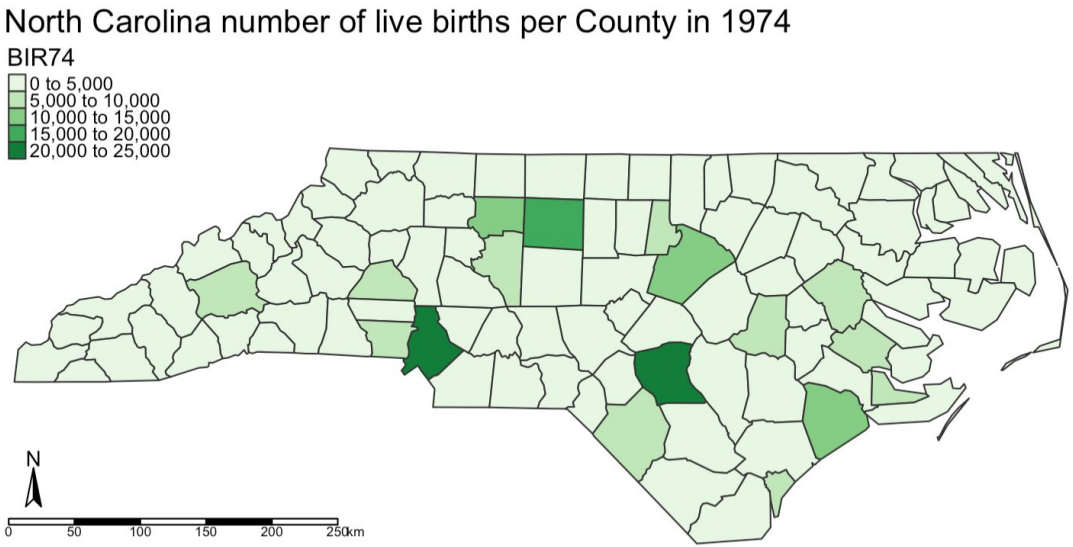
```r
# Solution: Adjust the bounding box to solve this!
bbox_nc <- st_bbox(nc)          # 1. Get the current bounding box
bbox_nc[4] <- bbox_nc[4] + 1    # 2. Adjust the ymax bounding box to desired height
bbox_nc <- st_as_sfc(bbox_nc) # 3. Convert the bounding box to an sf polygon

# Plot using tmap with the adjusted bounding box
tm_shape(nc, bbox = bbox_nc) +
  tm_borders() +
  tm_layout(title = "North Carolina Counties",
            frame = FALSE,
            title.size = 1.5)
```



tmap - Customization Progression 4

```
tm_shape(nc, bbox = bbox_nc) +
  tm_borders() +
  tm_fill(col = "BIR74") +
  tm_compass(type = "arrow", position = c("left", "bottom")) + # Add compass
  tm_scale_bar(position = c("left", "bottom")) +               # Add scale bar
  tm_layout(title = "North Carolina number of live births per County in 1974",
            frame = FALSE,
            title.size = 1.2)
```



tm_compass() Further Customization

- tm_compass(type = “___”...)
  - default: “arrow”
  - “4star”
  - “8star”
  - “radar”
  - “rose”

4star

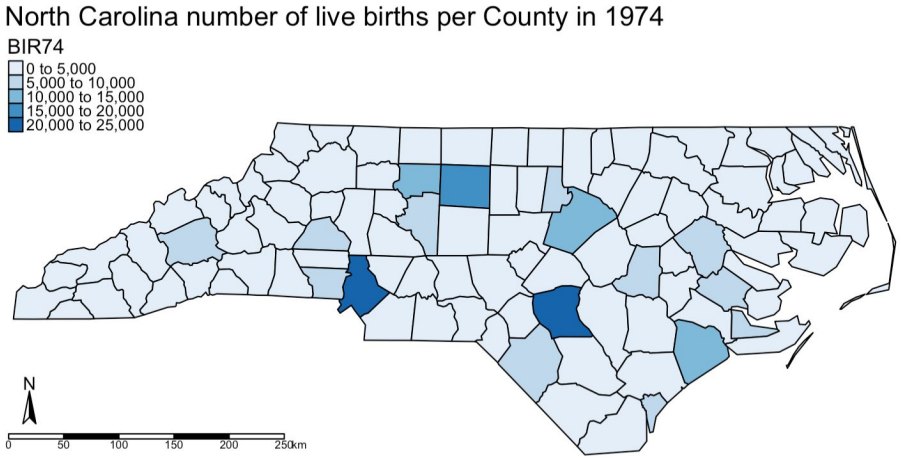
radar

8star

rose

tmap - Customization Progression 5

```r
# The default red color palette isn't very colorblind friendly, so lets use tmap_style()
tmap_style("col_blind")
tm_shape(nc, bbox = bbox_nc) +
  tm_borders() +
  tm_fill(col = "BIR74") +
  tm_compass(type = "arrow", position = c("left", "bottom")) + # Add compass
  tm_scale_bar(position = c("left", "bottom")) +               # Add scale bar
  tm_layout(title = "North Carolina number of live births per County in 1974",
            frame = FALSE,
            title.size = 1.2)
```



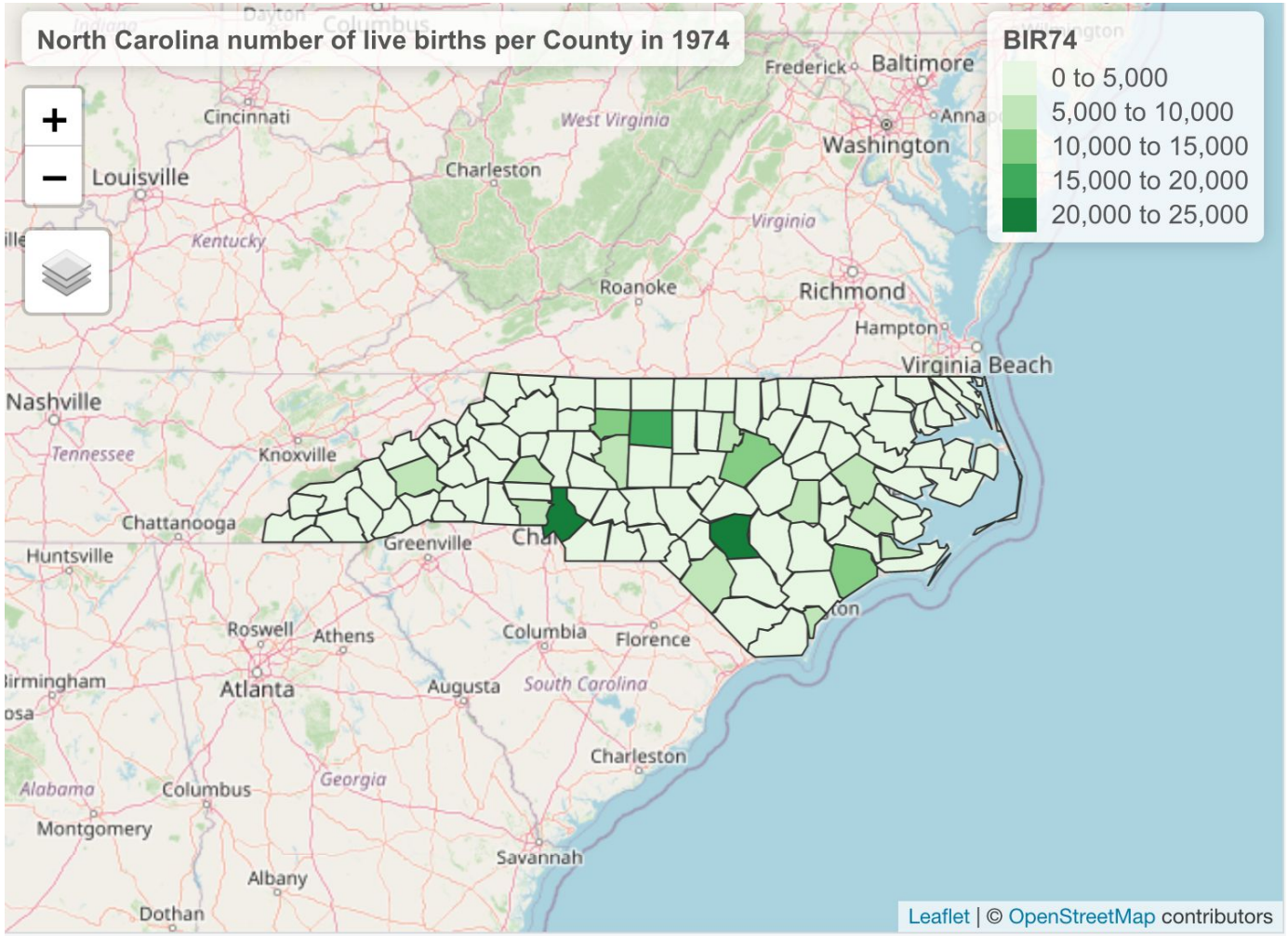
tmap - Interactive Maps

- **tmap_mode("view")** : will make any following maps interactive
- tmap_mode("plot") : will make any following maps static again
- tm_basemap() : allows users to call a basemap from online sources such as:
  - OSM
  - ESRI
  - Google

tmap - Interactive Maps Example

```r
# Lets make an interactive tmap!
tmap_mode("view")
tmap_style("watercolor")
tm_shape(nc, bbox = bbox_nc) +
  tm_borders() +
  tm_fill(col = "BIR74") +
  tm_basemap(server = "OpenStreetMap") + # You can adjust this provider basemap to your liking!
  tm_layout(title = "North Carolina number of live births per County in 1974",
            frame = FALSE,
            title.size = 1.2)
```

- Note: The interactive maps don’t play well with the compass and scale bar.

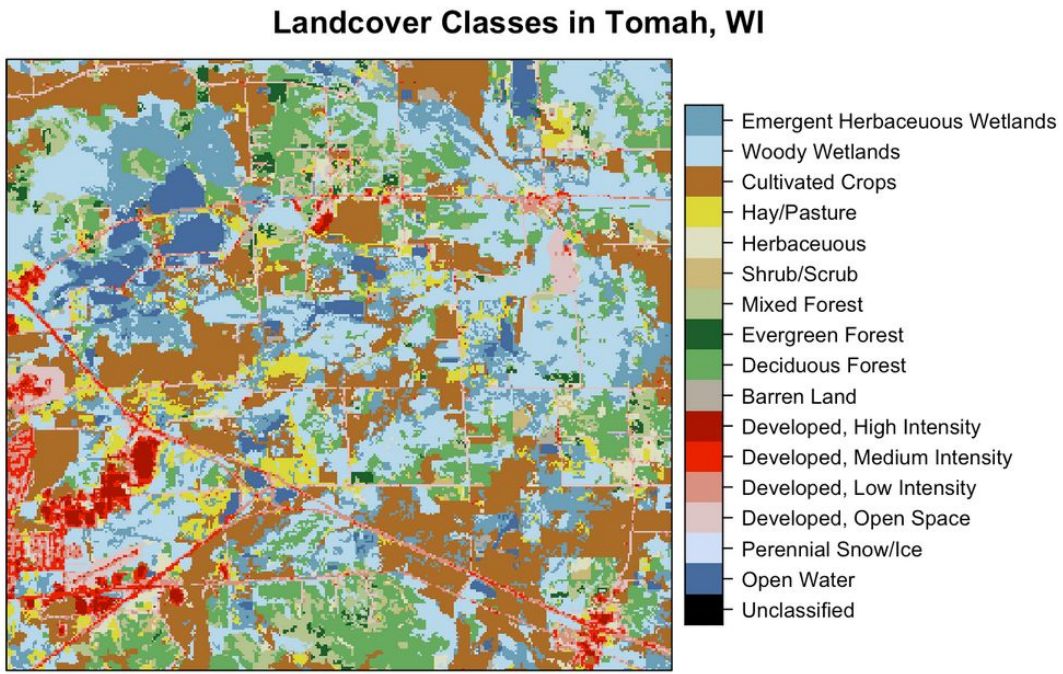

North Carolina number of live births per County in 1974
BIR74
0 to 5,000
5,000 to 10,000
10,000 to 15,000
15,000 to 20,000
20,000 to 25,000
Frederick
Baltimore
Washington
Cincinnati
West Virginia
Louisville
Charleston
Kentucky
Virginia
Roanoke
Richmond
Hampton
Virginia Beach
Nashville
Tennessee
Knoxville
Chattanooga
Huntsville
Greenville
Roswell
Athens
Columbia
Florence
Atlanta
Augusta
South Carolina
Charleston
Alabama
Columbus
Georgia
Montgomery
Savannah
Albany
Dothan
Leaflet | © OpenStreetMap contributors

3. What about rasters?

ggplot & tmap’s weird plotting relationship with rasters

- ggplot is not great with rasters…
- The aesthetics can be weird, and take a lot of work to get to a decent place
  - You won’t always want to use ggplot to make a pretty raster map unfortunately
- **QGIS** is much more straightforward as a free option to map
- Alternatively the **rasterVis** package is your best option in R

```r
# Create a plot with the original colors and correctly labeled landcover types
require(rasterVis)
levelplot(tomah_f,
          col.regions = col_tomah,
          main = "Landcover Classes in Tomah, WI",
          scales = list(draw = F))
```



4. Package Documentation

You will be reading documentation your whole career!

- R documentation is always changing as packages get updated
- Packages come and go, so you won’t always have me to show you how to use them!
- When learning to do any new analysis in R, it always starts with finding the package and learning how to use it.
- Learning how to use a package is as simple as reading the reference manual for the package.

Documentation Background

- “CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.”
- CRAN hosts all of the packages we’ve downloaded in R so far.
- All CRAN packages must have standardized documentation!
- If you’re looking for documentation on a specific function, you have options.
  1. CRAN Reference Manual
  2. RDocumentation.org
  3. rdrr.io
  4. Package-Specific Site
     - Example: https://dplyr.tidyverse.org/reference/

CRAN - Package rasterVis

https://cran.r-project.org/web/packages/rasterVis/index.html

rasterVis: Visualization Methods for Raster Data

Methods for enhanced visualization and interaction with raster data. It implements visualization methods for quantitative data and categorical data, both for univariate and multivariate rasters. It also provides methods to display spatiotemporal rasters, and vector fields. See the website for examples.

Version:	0.51.6
Depends:	R (≥ 4.0.0), methods, lattice (≥ 0.22-5)
Imports:	raster (≥ 3.4-13), terra (≥ 1.7-17), latticeExtra, stats, utils, parallel, grid, grDevices, RColorBrewer, hexbin, sp (≥ 1.0-6), zoo, viridisLite
Suggests:	rgl, ggplot2, colorspace, dichromat, sf
Published:	2023-11-01
Author:	Oscar Perpinan Lamigueiro [cre, aut], Robert Hijmans [aut], Alexandre Courtiol [ctb]
Maintainer:	Oscar Perpinan Lamigueiro <oscar.perpinan at upm.es>
BugReports:	https://github.com/oscarperpinan/rastervis/issues
License:	GPL-3
URL:	https://oscarperpinan.github.io/rastervis/
NeedsCompilation:	no
Citation:	rasterVis citation info
Materials:	README
In views:	Spatial, SpatioTemporal
CRAN checks:	rasterVis results

HERE!

Documentation:

Reference manual: rasterVis.pdf

Downloads:

Package source:	rasterVis_0.51.6.tar.gz
Windows binaries:	r-devel: rasterVis_0.51.6.zip, r-release: rasterVis_0.51.6.zip, r-oldrel: rasterVis_0.51.6.zip
macOS binaries:	r-release (arm64): rasterVis_0.51.6.tgz, r-oldrel (arm64): rasterVis_0.51.6.tgz, r-release (x86_64): rasterVis_0.51.6.tgz
Old sources:	rasterVis archive

Reverse dependencies:

Reverse depends:	ecochange
Reverse imports:	cati, cmsafvis, eRTG3D, geomod, gfcanalysis, IsoriX, lulcc, steps, stfit, TCHazaRds
Reverse suggests:	belg, ENMeval, isocat, LSRS, MetaLandSim, meteoForecast, PointedSDMs, raster, rasterdiv, rasterDT, SDMtune, solaR

Linking:

Please use the canonical form `https://CRAN.R-project.org/package=rasterVis` to link to this page.

lattice (version 0.10-10)

levelplot: Level Plots

Description

Draw Level Plots and Contour plots.

Usage

```
levelplot(formula, data,
          at,
          contour = FALSE,
          cuts = 15,
          pretty = FALSE,
          region = TRUE,
          ...,
          col.regions = trellis.par.get("regions")$col,
          colorkey = region)
contourplot(formula, data, at,
            contour = TRUE,
            labels = format(at),
            cuts = 7,
            pretty = TRUE,
            ...)
```

Arguments

`formula`	a formula of the form `` `z ~ x * y | g1 * g2 * ...` ``, where `` `z` `` is a numeric response, and `` `x, y` `` are numeric values evaluated on a rectangular grid. `` `g1,g2,...` `` are optional conditional variables, and
`data`	optional data frame in which variables are to be evaluated
`at`	numeric vector giving breaks along the range of `` `z` ``. Contours (if any) will be drawn at these heights, and the regions in between would be colored using `` `col.regions` ``.
`col.regions`	color vector to be used if regions is TRUE. The general idea is that this should be a color vector of moderately large length (longer than the number of regions. By default this is 100). It is expected that this vector would be gradually varyl
`colorkey`	logical specifying whether a color key is to be drawn alongside the plot, or a list describing the color key. The list may contain the following components:

rasterVis
Visualization Methods for Raster Data
Package index

Search the rasterVis package

Vignettes
Package overview
README.md

Functions 131

Source code 22

Man pages 17

bwplot-methods: Box and whisker plots of R
chooseRegion: Interaction with trellis objects
densityplot-methods: Density plots for Raste
gplot-methods: Use ggplot to plot a Raster*
hexbinplot: Formula methods
histogram-methods: Histogram of Raster ob
horizonplot-methods: Horizon plots of Raste
hovmoller-methods: Hovmoller plots
levelplot-methods: Level and contour plots o
miscLattice: Lines, points and polygons
plot3d: Interactive 3D plot of a RasterLayer
rasterTheme: Themes for 'raster' with 'lattice
rasterVis-package: Visualization Methods fo
splom-methods: Scatter plot matrices of Ras
vectorplot: Vector plots of Raster objects.
xyLayer: xyLayer
xyplot-methods: xyplot for Raster objects
Browse all...

Home / CRAN / rasterVis / **levelplot-methods: Level and contour plots of Raster objects.**

levelplot-methods: Level and contour plots of Raster objects.

In rasterVis: Visualization Methods for Raster Data

levelplot-methods R Documentation

Level and contour plots of Raster objects.

Description

Level and contour plots of Raster objects with `lattice` methods and marginal plots with `grid` objects.

Usage

```
## S4 method for signature 'Raster,missing'
levelplot(x, data = NULL, layers,
          margin = list(),
          maxpixels = 1e5,
          par.settings = rasterTheme(),
          between = list(x=0.5, y=0.2),
          as.table = TRUE,
```

Table of Contents

rasterVis

Introduction

The `rasterVis` package complements the `raster` and the `terra` packages, providing a set of methods for enhanced visualization and interaction. It defines visualization methods for quantitative data and categorical data, with `levelplot`, both for univariate and multivariate rasters.

It also includes several methods in the frame of the Exploratory Data Analysis approach: scatterplots with `xyplot`, histograms and density plots with `histogram` and `densityplot`, violin and boxplots with `bwplot`, and a matrix of scatterplots with `splom`.

On the other hand, `rasterVis` provides three methods to display spatiotemporal rasters: `hovmoller` produces Hovmöller diagrams, `horizonplot` creates horizon graphs, with many time series displayed in parallel, and `xyplot` displays conventional time series plots extracted from a multilayer raster.

Finally, this package is able to display vector fields using arrows, `vectorplot`, or with streamlines, `streamplot`.

This webpage illustrates some functionalities with examples. If you need more information, you may be interested in my book "**Displaying Time Series, Spatial, and Space-Time Data with R**". It includes four chapters devoted to the visualization of spatial and spatiotemporal raster data. Along with the main graphics from the text, its website offers access to the datasets used in the examples as well as the full R code.

Installation

The stable release of `rasterVis` can be found at CRAN. The development version is at GitHub.

Install the stable version with:

```
install.packages('rasterVis')
```

You can install the development version with the remotes package:

```
remotes::install_github('oscarperpinan/rasterVis')
```

or with the devtools package:

```
devtools::install_github('oscarperpinan/rasterVis')
```

Homework

1. Create a ggplot or tmap of both a point and polygon layer of your interest, and include:
   - Title, x-label, y-label
   - Custom color scheme
   - N Arrow + Scale Bar
2. Utilize what you know about reading documentation to plot a raster of your choice with rasterVis.

Deep Breath

I know this is a lot, but

R plotting takes practice