Lecture 10: Spatial Data in R

Raster Data

James Sears
AFRE 891 SS 24
Michigan State University

*Parts of these slides are adapted from **"R as GIS for Economists"** by Taro Mieno and **"R Geospatial Fundamentals"** by the UC Berkeley D-Lab used under **CC BY-NC-SA 4.0**.

Table of Contents: Today

Part 3: Raster Data

Prologue

To start, load in some packages:

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(prism, sf, terra, tidyterra,
               tictoc, tidyverse, tmap, units)
```

Preamble

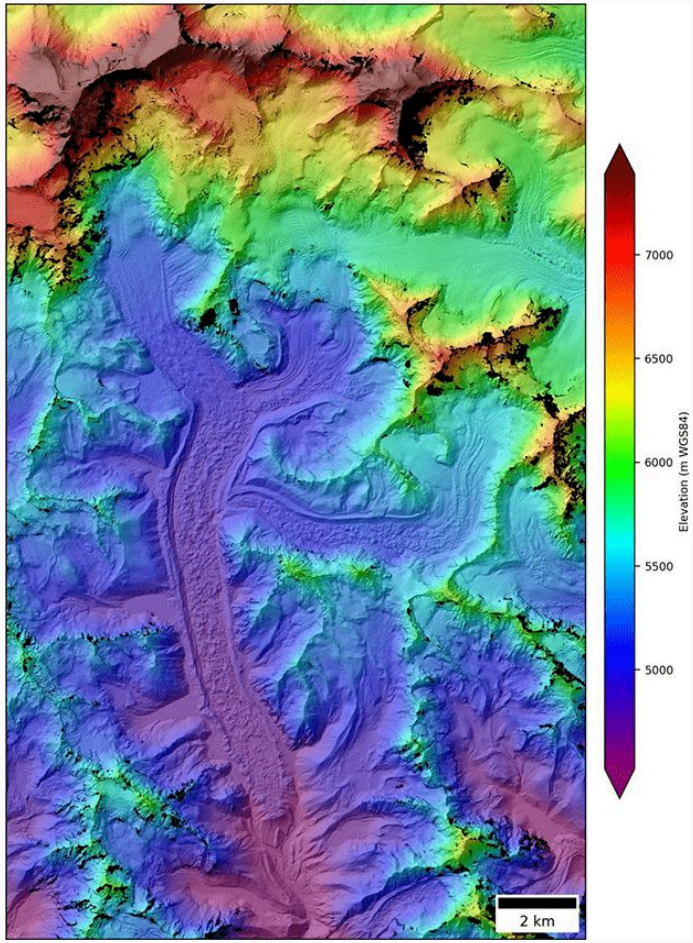
Let's also define a custom ggplot map theme

```
# add ggplot map theme
maptheme ← theme(
  panel.background = element_rect(fill = NA),
  # panel.border = element_rect(fill = NA, color = "grey75"),
  axis.text.x=element_blank(), #remove x axis labels
  axis.ticks.x=element_blank(), #remove x axis ticks
  axis.text.y=element_blank(),  #remove y axis labels
  axis.ticks.y=element_blank(),  #remove y axis ticks
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  legend.key = element_blank())
```

Intro to Raster Data

Raster Data: Regular Grids

Rasters are a way to represent **spatially continuous data** in a regular, **gridded format**



Raster Data: Regular Grids

Every raster is a **matrix of cells (pixels)** organized into a **grid**

- Covered area is divided into equal-area cells
- Grid size varies from raster to raster
- Each cell contains a **value**

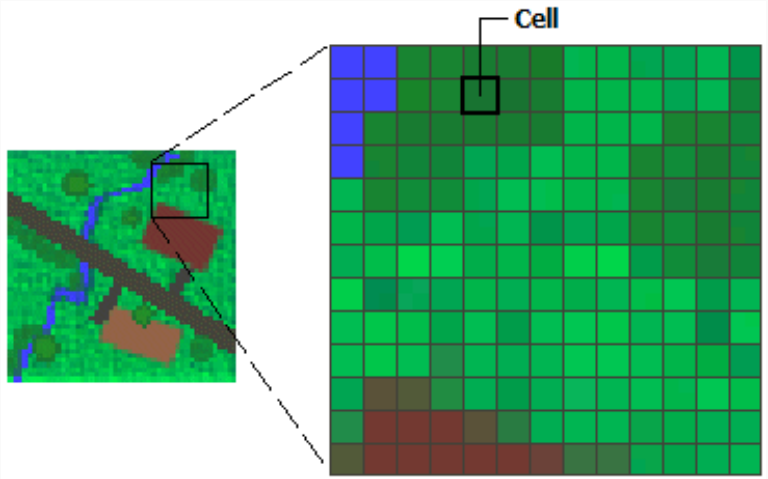


Image Source: ESRI

Raster Data: Regular Grids

Every raster is a **matrix of cells (pixels)** organized into a **grid**

Rasters can represent

- **Categorical data:** land use, soil type
- **Continuous phenomena:** temperature, elevation, spectral data from satellite images
- **Pictures:** scanned maps, drawings, photographs

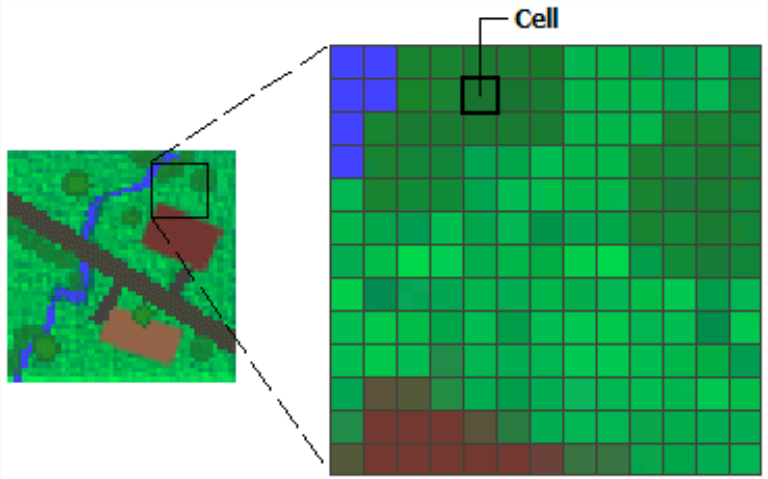
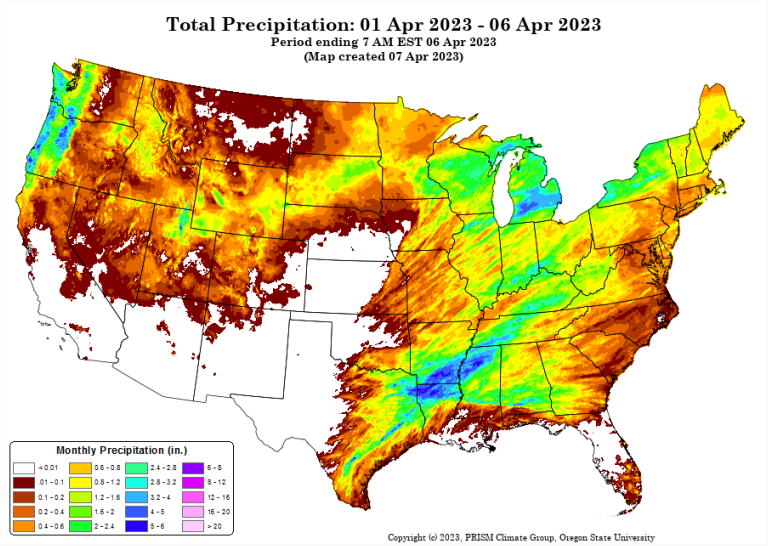


Image Source: ESRI

Single vs. Multi-Band

Raster data can come in **single** and **multi** band formats

- **Single-Band:** one band (layer) with a set of values
  - 1 value per grid cell
  - Ex: elevation, daily max temperature, land use



Single vs. Multi-Band

Raster data can come in **single** and **multi** band formats

- **Multi-band** multiple bands/layers per file
  - 1 value per grid cell **per layer**
  - Ex: MODIS satellite imagery - R, G, B, aerosol optical depth, reflectance, vegetation index, etc.

Landsat 8-9 Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS)

Bands	Wavelength (micrometers)	Resolution (meters)
Band 1 - Coastal aerosol	0.43-0.45	30
Band 2 - Blue	0.45-0.51	30
Band 3 - Green	0.53-0.59	30
Band 4 - Red	0.64-0.67	30
Band 5 - Near Infrared (NIR)	0.85-0.88	30
Band 6 - SWIR 1	1.57-1.65	30
Band 7 - SWIR 2	2.11-2.29	30
Band 8 - Panchromatic	0.50-0.68	15
Band 9 - Cirrus	1.36-1.38	30
Band 10 - Thermal Infrared (TIRS) 1	10.6-11.19	100
Band 11 - Thermal Infrared (TIRS) 2	11.50-12.51	100

USGS

Image Source: USGS

Common Raster Data

Common Raster Data

Raster data come in many different flavors

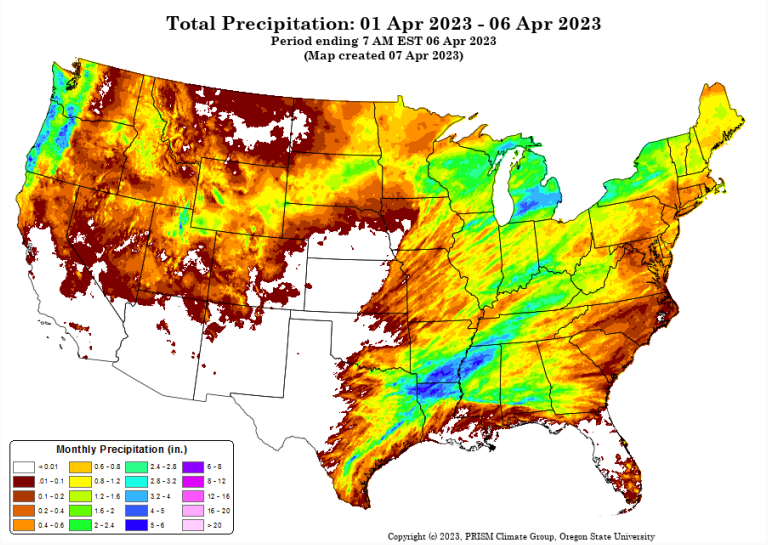
A few commonly used in Env, Ag, and Dev economics are

1. PRISM Gridded Climate Data
2. Nighttime Lights (NOAA, NASA, DoD)
3. National Land Cover Database (NLCD)
4. Cropland Data Layer (USDA)
5. Landsat (USGS, NASA)

PRISM

PRISM Climate Group produces several different climate products, including the **AN81d daily spatial climate dataset**

- 4km by 4km (or 800m by 800m) grid cells for CONUS, daily since Jan 1, 1981
- Precip, max/min/mean temp, mean dew point temp, min/max vapor pressure deficit
- Takes into account elevation, coastal effects, inversions, terrain barriers
- Standard product is available for free (800m grid available **for a fee**)
- Easily accessed directly in R with the `PRISM` package



Nighttime Lights

Currently produced under a NASA/NOAA partnership, began with Dept of Defense

Nighttime Lights

Light Every Night - World Bank Nighttime Light Data contains a complete archive of all daily nighttime imagery from the last 30 years.

- Analysis-ready, compiled from **two satellite imaging sources**
  1. Defense Meteorological Satellite Program Operational Linescan System (DMSP - OLS) , 1993- 2017
  2. Visible Infrared Imaging Radiometer Suite (VIIRS)
- **Open Nighttime Lights** has tutorials for accessing/performing operations and analysis with Night Lights data

Nighttime Lights

Light Every Night - World Bank Nighttime Light Data contains a complete archive of all daily nighttime imagery from the last 30 years.

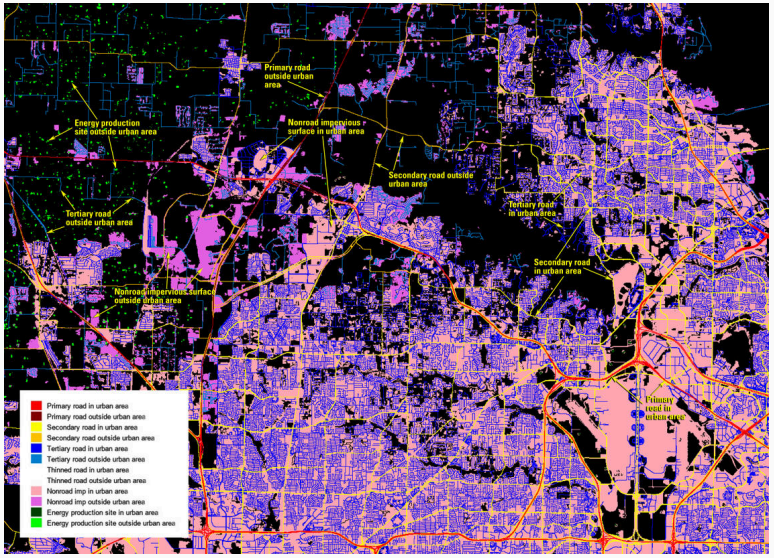
Other versions of the products exist

- DMSP or VIIRS Directly from the source
  - Cloud-free composites from **CO School of Mine's Earth Observation Group**
- **Harmonized version 1992-2018** from **Li et al.**

National Land Cover Database (NLCD)

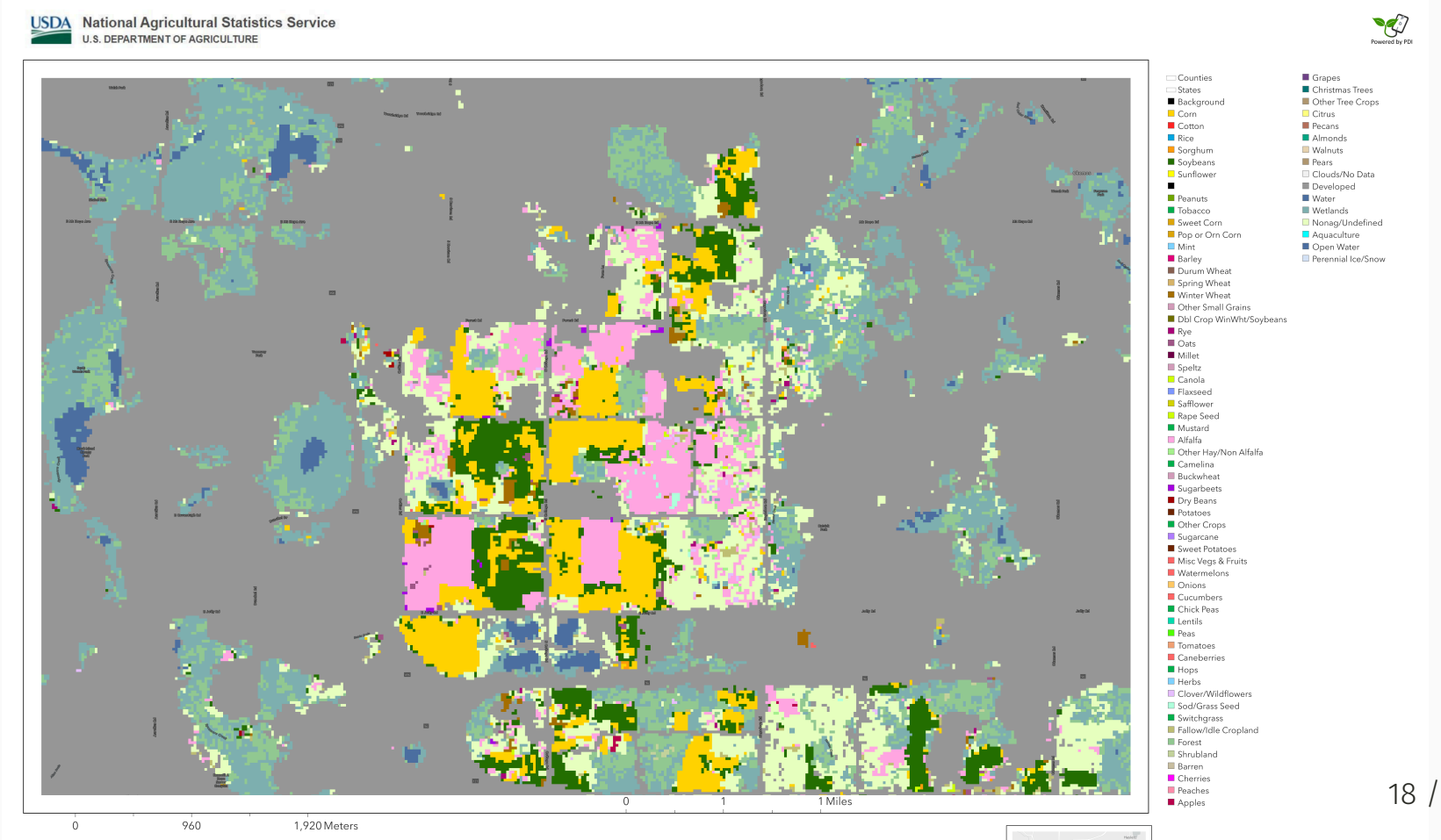
National Land Cover Database (NLCD) characterizes land cover and land cover change over time

- Land cover: water, ice, developed open/low/medium/high intensity, barren, forest, shrubland, herbaceous, planted/cultivated, wetlands
- Impervious surface descriptor categorizes roads, core urban areas, energy production sites



NASS Cropland Data Layer (CDL)

USDA's National Agricultural Statistics Service (NASS) produces the **Cropland Data Layer (CDL)** to **classify crop types**



NASS Cropland Data Layer (CDL)

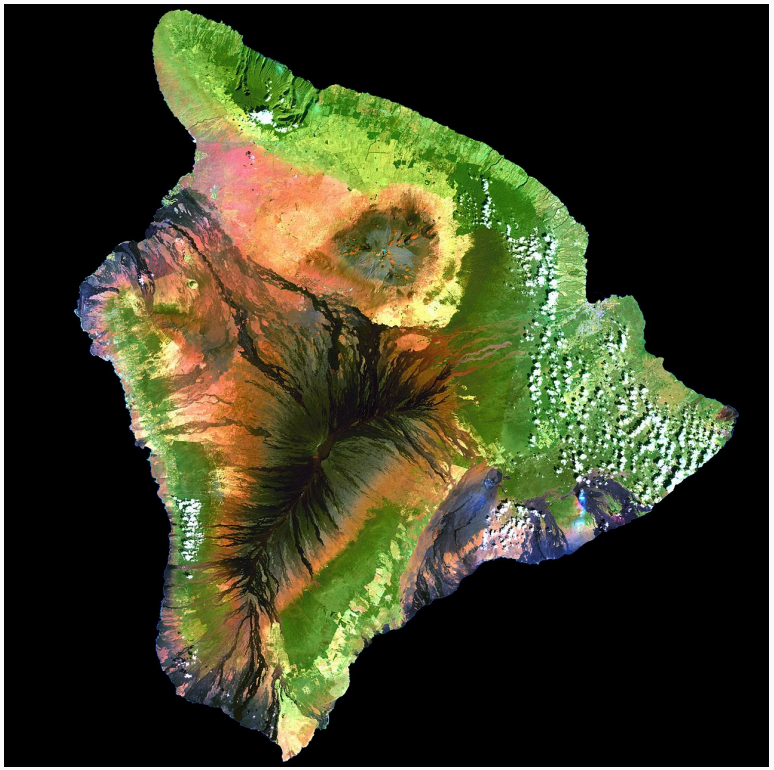
USDA's National Agricultural Statistics Service (NASS) produces the **Cropland Data Layer (CDL)** to **classify crop types**

- Works relatively well in areas with **few distinct crops** (i.e. corn and soy land)
- Less effective in areas with **lots of specialty crops** (i.e California, part of Michigan)
  - Hard to tell the difference between Red Russian kale and kohlrabi from a satellite
- Used as inputs to products like the **Corn and Soy Data Layer (CSDL)** that fill in areas/periods missing from CDL **(Wang et al.)**

Landsat

Landsat is the longest-running earth imaging project from NASA and USGS

- High-resolution, satellite images across many bands
  - Landsat 1-3 (1978-1983): Green, Red, Near Infrared (NIR), 60m resolution
  - Landsat 4-5 (1982 - 2013): add Blue, Shortwave IR, thermal, 30m res.
  - Landsat 6: didn't make it to orbit (RIP)
  - Landsat 7 (1999 - 2022): add Panchromatic, 15m res.
  - Landsat 8-9 (2013 -): add ultrablue, cirrus, more thermal



Other Raster Data Sources

Many other raster data products exist, including

- Digital Elevation Models (DEM) like the **National Elevation Dataset (NED)**
- **MODIS/Terra Vegetation Indices**
- **National Agriculture Imagery Program**
- **Nighttime Boat Detection using VIIRS**

Many handy collections of raster and other GIS data

- **NASA EarthData Collection**
- **USGS Collection**
- **US Forest Service**
- **Socioeconomic Data and Applications Center (SEDAC)**
- **Boston College List of Open Access GIS Data**
- **Michigan GIS Open Data**

Raster Operations

Raster Math (Map Algebra)

Map algebra uses math-like operations to perform operations **on rasters**

Largely fall into four camps:

1. **Local:** perform operations on a **cell-by-cell basis**
    - i.e. calculate difference between max and min daily temp for each cell
2. **Global:** apply bulk change to **all cells in a raster**
    - i.e. Rescale to percent, add/subtract a constant

Raster Math (Map Algebra)

Map algebra uses math-like operations to perform operations **on rasters**

Largely fall into four camps:

1. **Focal:** create value using **neighboring cells' values**
    - i.e. average over current and contiguous cells
2. **Zonal:** apply math function to group of cells in a**specific zone**
    - i.e. total precipitation within a watershed

Raster Math (Map Algebra)

Map algebra uses math-like operations to perform operations **on rasters**

Other common raster operations include

- Cropping to extents of other rasters or vectors
- Extracting values from grid cells for overlayed vectors

Raster Packages in R

There are several packages we can and may want to use to manipulate rasters in R.

1. raster

- Most popular, longstanding way to handle raster data
- Many other econ-oriented packages were written for it

2. terra

- Newer package to replace `raster`
- Shares most function names with `raster`
- Written in C++, so much faster than `raster`

Raster Packages in R

There are several packages we can and may want to use to manipulate rasters in R.

1. stars

- Designed for handling spatiotemporal raster data
- Directly compatible with **sf** (same developer)
- Allows **dplyr** language (filter, mutate, etc.)

raster and terra

raster is the traditional way to read/load/interact with rasters

- Use `raster()` to read **single-band** rasters
  - Class `RasterLayer`
- Use `brick()` to read **multi-band** rasters
  - Class `RasterBrick`

terra has only one object class: `SpatRaster`

- no distinction between single and multi-layer rasters

Working with Rasters

Let's get some practice working with rasters, using the PRISM data.

We can use the **prism** package to download the files we want. Let's start by downloading two files: max and min daily temp for April 4, 2023

```
# Set the download path to the PRISM data folder
options(prism.path = "data/PRISM/")

# Download the data
get_prism_dailys(type='tmax',minDate='2023-04-04',maxDate='2023-04-04',ke
get_prism_dailys(type='tmin',minDate='2023-04-04',maxDate='2023-04-04',ke
```

Working with Rasters

Let's read in the single-band PRISM max daily temp data as a `SpatRaster` using **terra** methods

```
tmax ← rast("data/PRISM/PRISM_tmax_stable_4kmD2_20230404_bil/PRISM_tmax_s
```

Working with Rasters

Get a summary of the structure and contents just by calling the object's name

```
tmax
```

```
## class       : SpatRaster
## dimensions  : 621, 1405, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -125.0208, -66.47917, 24.0625, 49.9375  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat NAD83
## source      : PRISM_tmax_stable_4kmD2_20230404_bil.bil
## name        : PRISM_tmax_stable_4kmD2_20230404_bil
## min value   :                              -17.3296
## max value   :                               38.9204
```

Working with Rasters

Retrieve the CRS with `terra::crs()`

- Necessary when we want to interact with vector objects!

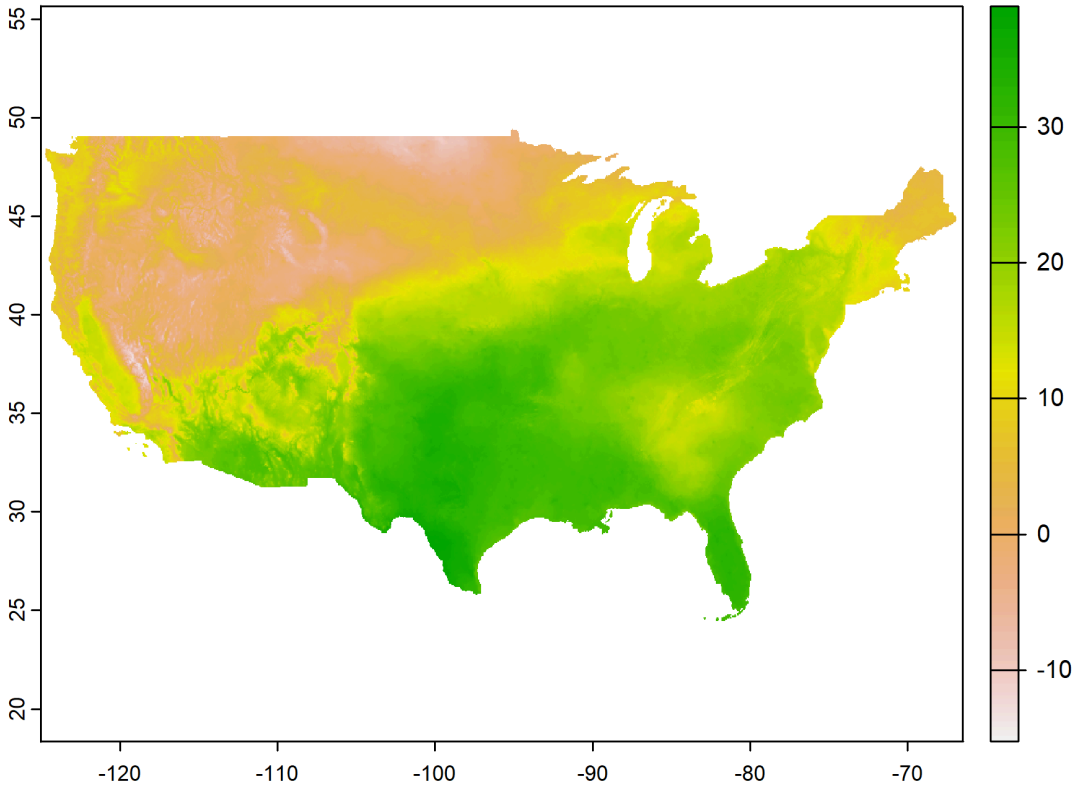
```
prism_crs ← terra::crs(tmax)
prism_crs
```

```
## [1] "GEOGCRS[\"NAD83\",\n    DATUM[\"North American Datum 1983\",\n        E
```

Working with Rasters

And plot the raster with the `plot()` function

```
plot(tmax)
```



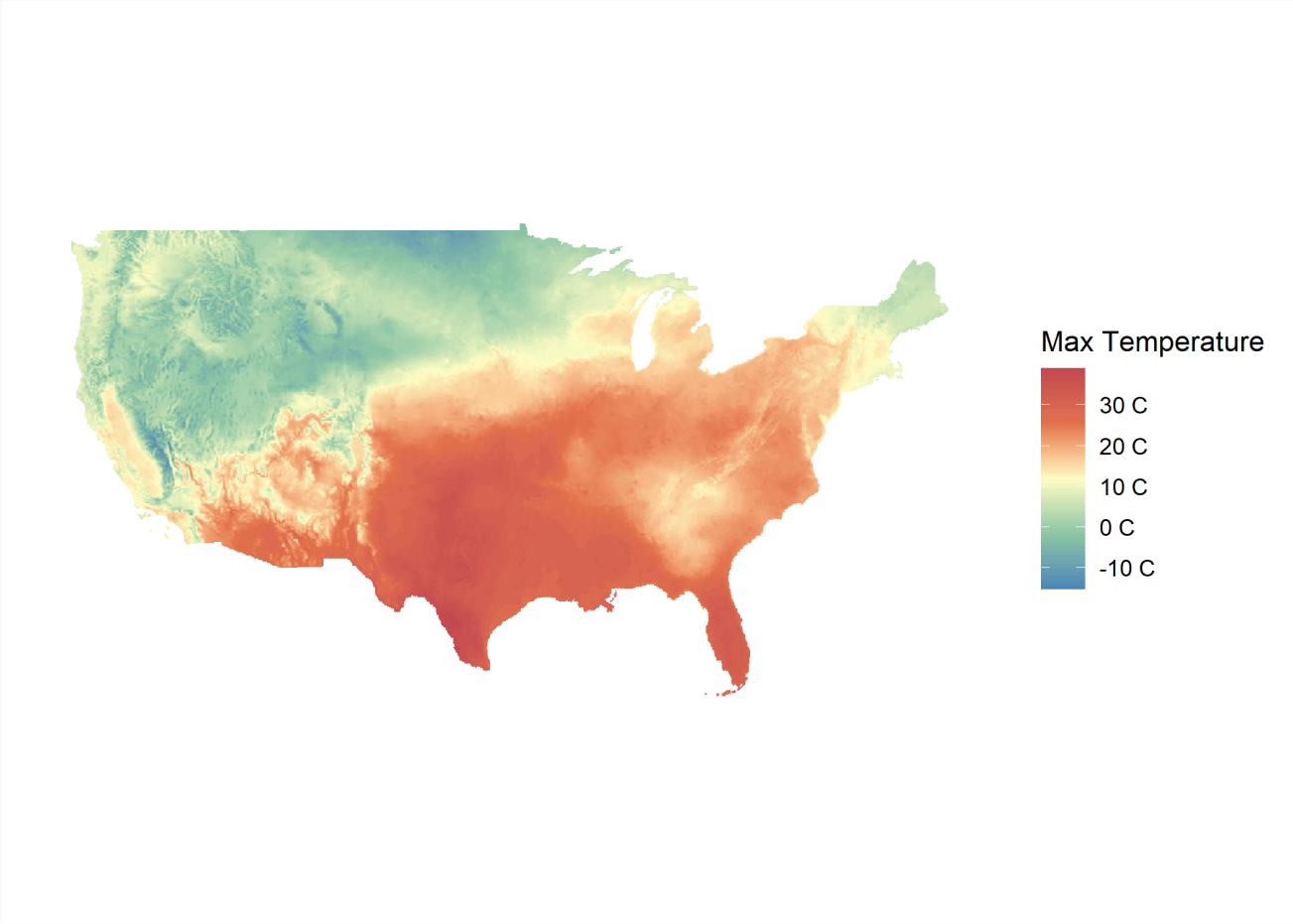
terra and ggplot2

To plot rasters in ggplot, use `geom_spatraster()` from the **tidyterra** package:

```
pacman::p_load(tidyterra)
ggplot() + geom_spatraster(data = tmax) +
  maptheme +
  scale_fill_whitebox_c(
    palette = "muted",
    labels = scales::label_number(suffix = " C")) +
  labs(fill = "Max Temperature")
```

terra and ggplot2

To plot `SpatRasters` in ggplot, use `geom_spatraster()` from the **tidyterra** package:



Working with Rasters

We can easily create a **multi-layer** `SpatRaster` object from two single-layer objects using `c()`

Let's load in the same day's minimum temp and create a high/low temp multi-layer raster

```
tmin ← rast("data/PRISM/PRISM_tmin_stable_4kmD2_20230404_bil/PRISM_tmin_stable_4k

high_low ← c(tmin, tmax)
high_low
```

```
## class       : SpatRaster
## dimensions  : 621, 1405, 2  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -125.0208, -66.47917, 24.0625, 49.9375  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat NAD83
## sources     : PRISM_tmin_stable_4kmD2_20230404_bil.bil
##               PRISM_tmax_stable_4kmD2_20230404_bil.bil
## names       : PRISM_tmin_stab~D2_20230404_bil, PRISM_tmax_stab~D2_20230404_bil
## min values  :                        -27.6181,                        -17.3296
## max values  :                         27.3304,                         38.9204
```

Multi-layer SpatRasters

Access specific layers with indexing `[[]]`

```
# retrieve the second layer only (tmax)
high_low[[2]]
```

```
## class       : SpatRaster
## dimensions  : 621, 1405, 1  (nrow, ncol, nlyr)
## resolution  : 0.04166667, 0.04166667  (x, y)
## extent      : -125.0208, -66.47917, 24.0625, 49.9375  (xmin, xmax, ymin, yma
## coord. ref. : lon/lat NAD83
## source      : PRISM_tmax_stable_4kmD2_20230404_bil.bil
## name        : PRISM_tmax_stable_4kmD2_20230404_bil
## min value   :                              -17.3296
## max value   :                               38.9204
```

Multi-layer SpatRasters

And access values using the `values` function (or try to...)

```
temp_vals ← terra::values(high_low)
head(temp_vals)
```

```
##      PRISM_tmin_stable_4kmD2_20230404_bil PRISM_tmax_stable_4kmD2_20230404_b
## [1,]                                 NaN                                  N
## [2,]                                 NaN                                  N
## [3,]                                 NaN                                  N
## [4,]                                 NaN                                  N
## [5,]                                 NaN                                  N
## [6,]                                 NaN                                  N
```

Returns a matrix with

- One column per layer
- One row per cell

terra and Memory

Since raster images can be quite large, **terra** didn't actually store cell values **in memory** when we loaded the two rasters (hence the `NaN`s).

Instead, we established a **connection** to the file, allowing us to retrieve the values **only when necessary**.

terra and Vectors

terra also has its own vector object class: `SpatVector`

We can easily go from an `sf` object to `SpatVector` using `vect()`:

```
# get counties shapefile as an sf object
counties ← st_read("data/MichiganCounties/MichiganCounties.shp", quiet =

# convert to SpatVector
counties_vect ← counties %>%
  vect()

class(counties_vect)
```

```
## [1] "SpatVector"
## attr(,"package")
## [1] "terra"
```

raster

```
library(raster)
```

When we load in the **raster** package, it **masks** the `select` function

- Running `select()` now defaults to the version specific to raster data (the **raster** package version)
- To use the `dplyr` version, we now need to call it explicitly with `dplyr::select()`

Packages **warn you in the console** when masking occurs!

Convert Raster to sf

Historically, going from raster to sf took a few steps.

1. Use **raster** package to convert raster to polygons (`SpatialPolygons`)

2. Use **sf** to convert sp object to sf

Convert Raster to sf

Let's give it a try using PRISM data.

```
# first convert to spatialPolygons
tmax_vec ← raster::rasterToPolygons(tmax)
```

```
## Error in (function (classes, fdef, mtable) : unable to find an inherited met
```

Hmm, looks like `raster` functions don't play great with `SpatRaster` objects from **terra**.

Convert Raster to sf

We can use `as()` to easily go between `raster` and `terra` objects:

```
tmax_rast ← as(tmax, "Raster")
```

Now retry the polygon conversion (note: this takes a *while* to run)

```
tmax_vec ← raster::rasterToPolygons(tmax_rast)
```

Convert Raster to sf

Q: what did we get back?

```
class(tmax_vec)
```

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

To get it into an sf object, we can use `st_as_sf()` (again, this can take a while)

```
tmax_vec_sf ← st_as_sf(tmax_vec)
```

Convert Raster to sf

We now have an sf object, with

- One row per polygon
- A single attribute column
- Geometry column for the original grid cell

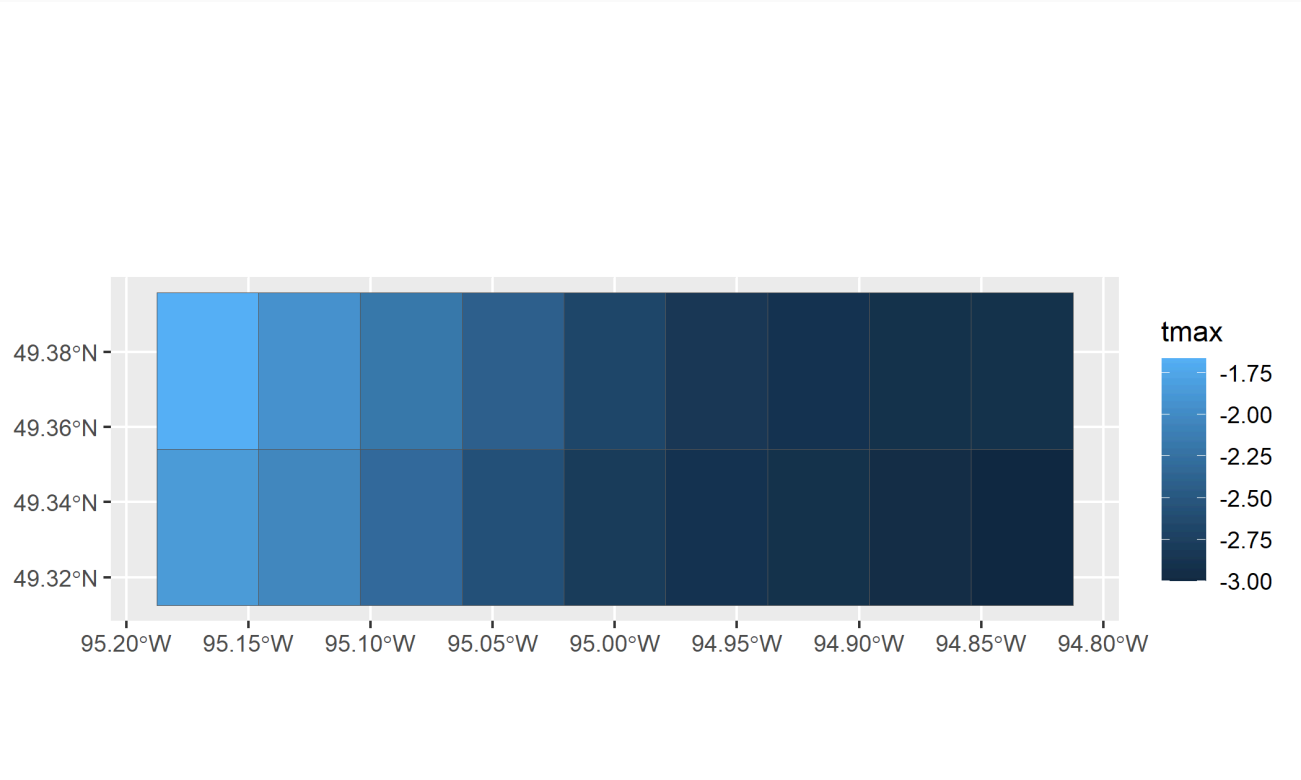
```
head(tmax_vec_sf)
```

```
## Simple feature collection with 6 features and 1 field
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: -95.1875 ymin: 49.35417 xmax: -94.97917 ymax: 49.4375
## CRS:           +proj=longlat +datum=NAD83 +no_defs
##   PRISM_tmax_stable_4kmD2_20230404_bil                       geometry
## 1                              -1.6414 POLYGON ((-95.14583 49.4375 ...
## 2                              -1.6634 POLYGON ((-95.1875 49.39583 ...
## 3                              -1.9247 POLYGON ((-95.14583 49.3958 ...
## 4                              -2.1725 POLYGON ((-95.10417 49.3958 ...
## 5                              -2.4084 POLYGON ((-95.0625 49.39583 ...
## 6                              -2.6685 POLYGON ((-95.02083 49.3958 ...
```

Convert Raster to sf

Plotting emphasizes that each polygon is the original grid cell:

```
colnames(tmax_vec_sf)[1] ← "tmax"
ggplot() +
  geom_sf(aes(fill = tmax), data = tmax_vec_sf[2:19,])
```



Convert Raster to sf

One nice thing about this process: the original CRS tags along the whole way through.

```
prism_crs
```

```
## [1] "GEOGCRS[\"NAD83\",\n    DATUM[\"North American Datum 1983\",\n        E
```

```
st_crs(tmax_vec_sf)
```

```
## Coordinate Reference System:
##   User input: +proj=longlat +datum=NAD83 +no_defs
##   wkt:
## GEOGCRS["unknown",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]],
##         ID["EPSG",6269]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433],
##         ID["EPSG",8901]],
```

Convert sf to Raster

We can more easily go the other direction with `st_rasterize()` from the **stars** package to rasterize an sf object.

```
county_rast ← dplyr::select(counties, SQMILES) %>%
  stars::st_rasterize()
```

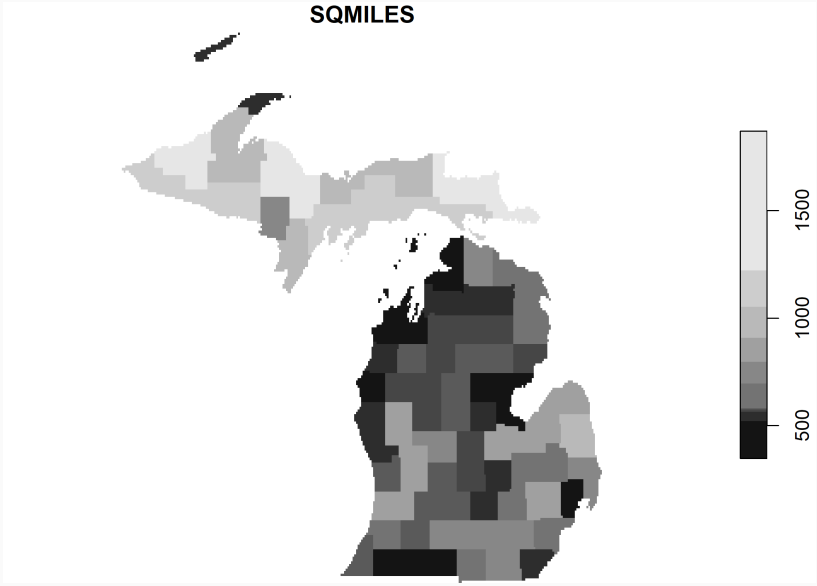
Convert sf to Raster

This process yields a `stars` object, which represents a dense spatiotemporal array

```
class(county_rast)
```

```
## [1] "stars"
```

```
plot(county_rast)
```



Combining Raster and Vector Data

Combining Raster and Vector Data

Often we want to **combine** raster and vector data

- Plot administrative boundaries over nightlights
- Extract vegetation indices for farm fields

I often do this to add variables for use in econometric models

Let's work through an example workflow of both for **extracting PRISM data**

CRS Agreement

As when working with multiple vector objects, we first need to make sure our raster and vector objects are on the **same CRS**

```
# read in the Wayne County census tracts
tracts ← st_read("data/wayne_acs", quiet = TRUE)

identical(st_crs(high_low), st_crs(tracts))
```

```
## [1] FALSE
```

```
# not the same, so transform tracts to prism CRS
tracts ← st_transform(tracts, crs = st_crs(high_low))

identical(st_crs(high_low), st_crs(tracts))
```

```
## [1] TRUE
```

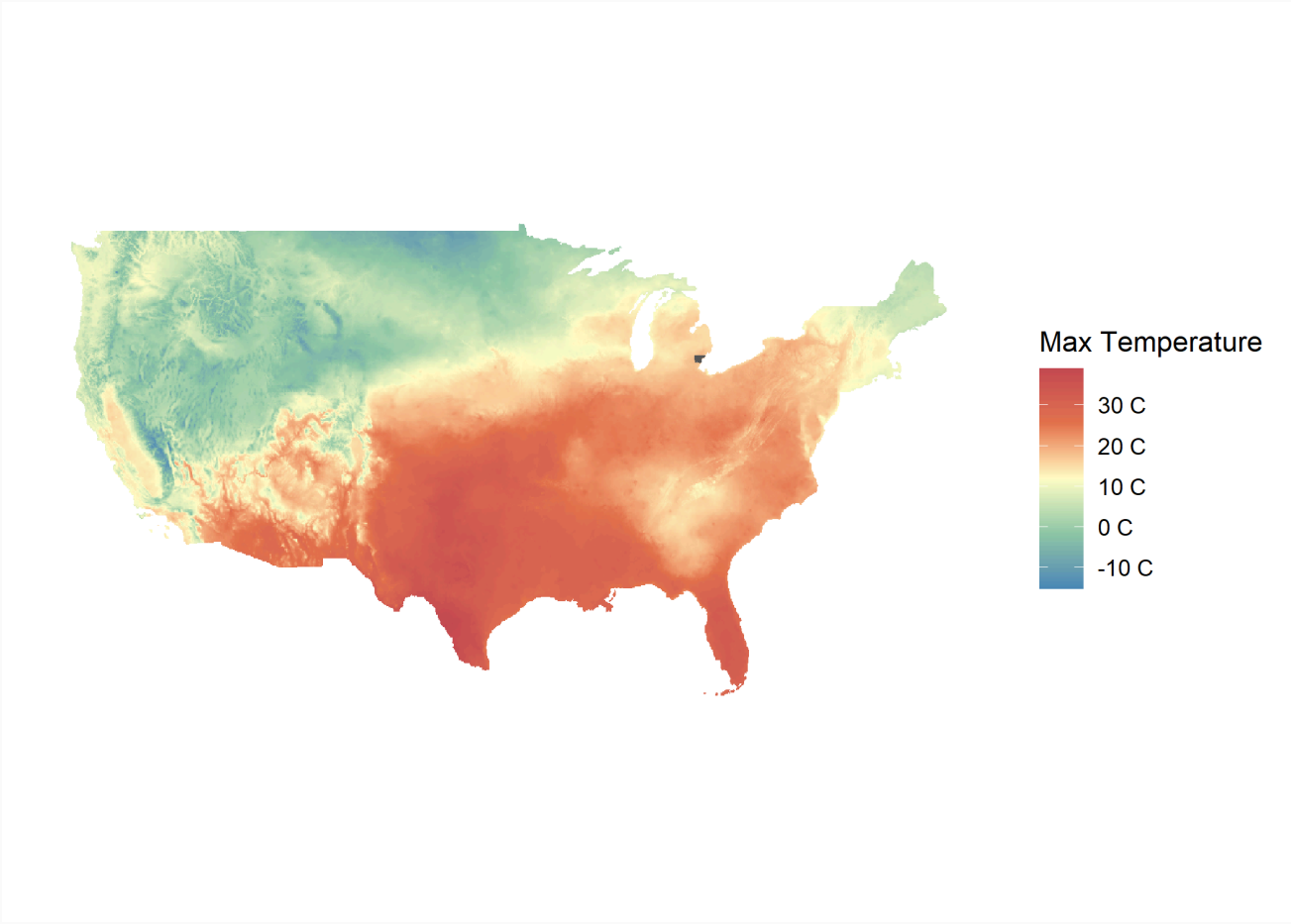
Plotting

Now that we have them on the same CRS, we can plot them together.

```
ggplot() + geom_spatraster(data = tmax) +
  geom_sf(data = tracts) +
  maptheme +
  scale_fill_whitebox_c(
    palette = "muted",
    labels = scales::label_number(suffix = " C")) +
  labs(fill = "Max Temperature")
```

Plotting

Now that we have them on the same CRS, we can plot them together.



Adjust the Viewing Area

That *technically* worked, but it defaulted to the widest extent of the two objects.

If we want to focus in on a specific area (i.e. Wayne County), we can crop the raster using the `crop()` function from `terra`.

First, let's figure out what the CRS units are for the census tracts:

```
st_crs(tracts) # view units of current CRS (meters)
```

```
## Coordinate Reference System:
##   User input: NAD83
##   wkt:
## GEOGCRS["NAD83",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]],
##         ID["EPSG",6269]],
##     PRIMEM["Greenwich",0,
```

Cropping

Now, crop to a 25mi buffer around Wayne county

- Converting to the right units (meters)!

```
# crop to Wayne County with 25mi buffer:
tmax_crop ← terra::crop(tmax, st_buffer(tracts, dist = 40233.6))

crop_plot ← ggplot() + geom_spatraster(data = tmax_crop) + geom_sf(data =
  maptheme + scale_fill_whitebox_c(palette = "muted",
    labels = scales::label_number(suffix = " C")) +
  labs(fill = "Max Temperature")
```

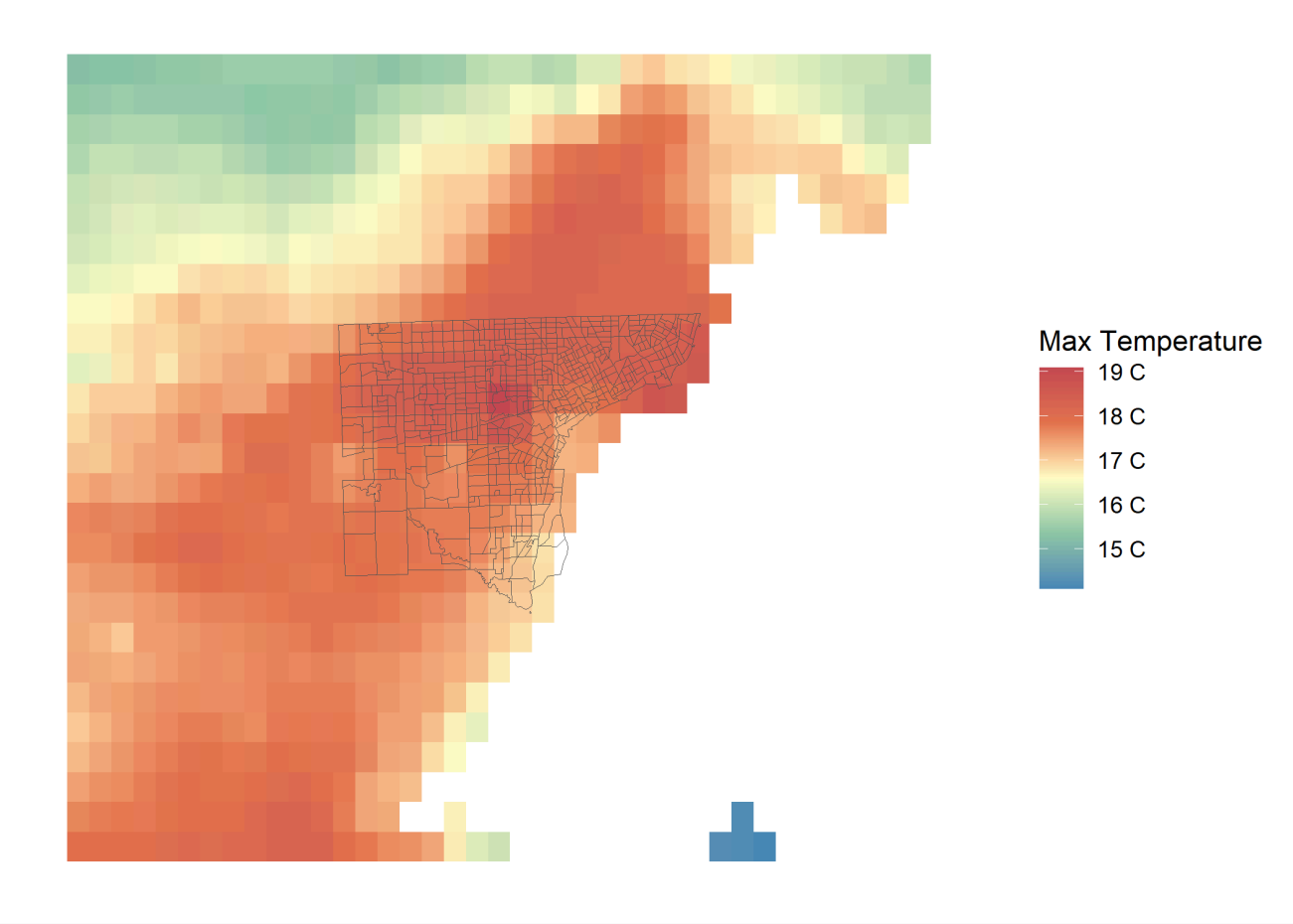
Cropping

And plotting:

```
crop_plot ← ggplot() + geom_spatraster(data = tmax_crop) + geom_sf(data =
  maptheme + scale_fill_whitebox_c(palette = "muted",
    labels = scales::label_number(suffix = " C")) +
  labs(fill = "Max Temperature")
```

1. Cropping

```
crop_plot
```



Extracting from Raster

Next, we can extract max temp values for each of our polygons using

- `raster`
  - slowest
- `terra`
  - moderate speed
- `velox`
  - super fast (orders of magnitude faster)
  - trickier to download/setup

Note that it's often best to extract from a **cropped raster** to reduce computational time if you have a big raster

Raster Extract

Extracting with `raster`:

```
tic()
extr_raster ← raster::extract(tmax_crop, tracts, df = TRUE)
toc(quiet = FALSE)
```

```
## 0.22 sec elapsed
```

Extract Output

Inspecting the output:

```
head(extr_raster, 10)
```

```
##    ID PRISM_tmax_stable_4kmD2_20230404_bil
## 1   1                                17.8561
## 2   2                                18.0926
## 3   2                                18.4322
## 4   2                                18.2992
## 5   2                                18.0926
## 6   3                                18.2117
## 7   3                                18.3637
## 8   3                                18.2117
## 9   4                                18.1615
## 10  4                                18.1856
```

Extract Output

```
dim(extr_raster)
```

```
## [1] 1485    2
```

Since we didn't specify a summary function, we got back a data frame with

- one row per intersecting grid cell
- `ID` column indicates the matching sf object's row
    - Third tract intersects three grid cells

Extract Output

Specify a summary function with the `fun` argument to get **one row per polygon**

- `df = TRUE` no longer used

```
extr_raster_mean ← raster::extract(tmax_crop, tracts, fun = mean)
head(extr_raster_mean)
```

```
##   ID PRISM_tmax_stable_4kmD2_20230404_bil
## 1  1                                17.85610
## 2  2                                18.22915
## 3  3                                18.26237
## 4  4                                18.16953
## 5  5                                18.39983
## 6  6                                17.74960
```

Terra Extract

Equivalently using terra's `extract`

```
tic()
extr_terra ← terra::extract(tmax_crop, tracts, fun = mean)
toc(quiet = FALSE)
```

```
## 0.25 sec elapsed
```

```
head(extr_terra)
```

```
##   ID PRISM_tmax_stable_4kmD2_20230404_bil
## 1  1                                17.85610
## 2  2                                18.22915
## 3  3                                18.26237
## 4  4                                18.16953
## 5  5                                18.39983
## 6  6                                17.74960
```

Velox Extract

The `velox` package can perform extractions a **lot faster** than `raster`

- 122.2x faster according to **their benchmarking**
- Fully interoperable with `sf` package
- Can extract for polygons, lines, and points
- Performs all raster operations in C++

This speed comes at a **slight cost**, though:

- Not currently on CRAN, so must install from Github (along with dependencies)
- Syntax doesn't match anything we've done so far
- Doesn't play nicely with `terra` objects (SpatRaster, SpatVector, etc.)
- Specific considerations for small polygons
  - By default returns NA if polygon doesn't intersect with at least one grid cell's centroid
  - Must use `small = TRUE` argument

Velox Installation

Since **velox** isn't on CRAN, we'll install it straight from GitHub

```
if (!require("velox")){
  # install dependencies
remotes::install_github('cran/rgeos')
remotes::install_github('cran/rgdal')

#then install velox
remotes::install_github('hunzikp/velox')
}
```

Velox Installation

Turns out that **rgdal**, which has been archived on CRAN, can't install directly from GitHub either.

Let's do it manually then from the **archive**.

- Note that you have to
  1. Be on an older version of R (4.2.0 works)
  2. Have Rtools installed (Windows) for *the older version of R* (i.e. 42)
     - Switch from `Tools > Global Options > R Version > Change`

```
# Download package tarball from CRAN archive
url ← "https://cran.r-project.org/src/contrib/Archive/rgdal/rgdal_1.6-7.

# Install package
install.packages(url, type="source", repos=NULL)
```

Velox Installation

Oof, that took some work - but now we can install velox:

```
if (!require("velox")){
  #install velox
  remotes::install_github('hunzikp/velox')
}
```

Velox Syntax

`velox` uses its own `VeloxRaster` object, which you can create from one of our existing rasters using `velox()`:

```
pacman::p_load(velox)
vx_tmax ← velox(tmax_crop)
```

```
## Error in velox(tmax_crop): x is not of a supported class.
```

- RasterLayer, RasterStack, or RasterBrick (depending on the number of layers)

Velox Syntax

We can use `as()` to easily go between `raster` and `terra` objects:

```
tmax_crop_rast ← as(tmax_crop, "Raster")
```

Now that we have our Raster object, we can create the `VeloxRaster` object using `velox()`:

```
vx_tmax ← velox(tmax_crop_rast)
class(vx_tmax)
```

```
## [1] "VeloxRaster"
## attr(,"package")
## [1] "velox"
```

You can go back to Rasters with `stack ← vx$as.RasterStack()`

Velox Extract

Now we can extract with `VeloxRaster$extract(sf, fun, small, df)`

- `sf` the sf object to extract for
- `fun` the function to use
  - If omitted, returns the value for every matching grid cell
- `small = TRUE` to return values when polygons don't intersect grid cell centroids
- `df = TRUE` to return a dataframe (otherwise returns a list)

```
tic()
extr_velox ← vx_tmax$extract(tracts, fun = mean, small = TRUE, df = TRUE
toc(quiet = FALSE)
```

```
## 0.03 sec elapsed
```

0.02 seconds! While it took more work to get **velox** working, the fixed costs are worth it for large extraction tasks. An easier alternative: **exactextractr**

Challenge

Using the same datasets we've used in these (and previous) slides:

1. **Construct a time series of county-level total annual precipitation for Michigan Counties**

2. **How has the share of annual precipitation falling outside of the corn growing season changed over the last 20 years?**

    - Use monthly long-term precip data from **PRISM LT81m**

Write out a step-by-step plan, then implement it.

Table of Contents: Today