

# RStudio for Practical Geospatial Analysis

Roger Andre, Solutions Engineer, Posit PBC

# Practical Geospatial Analysis...

‘Practical’ as in, “*We need to know...*”

- *How many people live in a service area?*
- *How far apart are 2 businesses from one another?*
- *What are the geographic coordinates for an address?*
- *How many businesses are within N miles of an address?*

“*...by tomorrow!*”

# R widely used in Data Science but...

*...using R/RStudio for spatial analysis still isn't very common.*

- Spatial Data Analysis (GIS) in R is still considered “niche” by some people
- Older spatial packages (like `sp`) implemented very R-centric workflows
- There are PLENTY of other, good GIS tools available

*My goal here is to convince you to try it!*

# What is R?

*“R is a free software environment for statistical computing and graphics.”*

- An interpreted language that is similar in some ways to Python
- Multipurpose, but highly optimized for working with structured data
- Pandas library in Python is based on core functionality in R

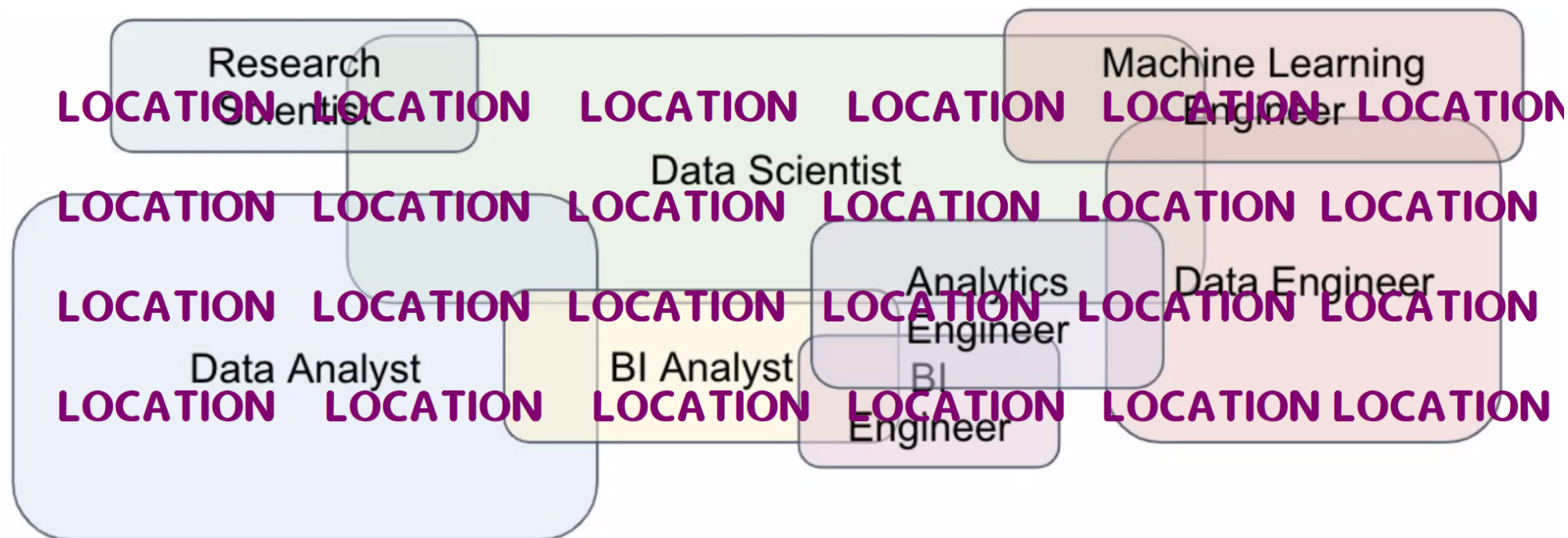
# What is RStudio?

- Open Source IDE with tools for working in R (and Python)
- Available in free (as in speech) and commercial editions
- Desktop app. on many OS and server application on Linux

**RStudio** was also formerly the name of the company that funds work on the IDE (and other OS projects).

# Why should you use R/RStudio for Geospatial Data Analysis?

*Because you're probably already using RStudio for ~~Geospatial~~ Data Analysis!*



# And since you're already using RStudio...

*RStudio has built-in features that are well suited to GIS!*

# Native graphic support for vector “maps”

Charts, plots, images...



# ...and raster plots

## Gridded Population

The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for creating a gridded population raster plot. The code defines a bounding box (`wa_bbox`), reads a population raster (`wa_pop`), and plots it with a legend. The legend text is set to "Population".
- Environment:** Shows the current environment with variables:
 

Variable	Class	Details
states	Formal class	52 obs. of 10 variables
wa	Formal class	1 obs. of 10 variables
wa_pop	Formal class	RasterLayer
world_pop	Formal class	RasterLayer
- Plots:** Shows a map of Washington state with a population density overlay. The legend indicates population values from 0 to 7000.
- Console:** Shows the execution of the R code, including the creation of the raster and the plotting of the map.

# ...and other things

(my slides under development)

The image shows a screenshot of the RStudio interface with a Quarto presentation slide displayed. The slide content is:

## Native graphic support for “maps”

Charts, plots, images...

8 / 12

The background of the slide shows RStudio code and console output. The code in the source pane includes:

```

73 - ## And since you're already using RStudio...
74 - . . .
75 - </br>
76 -
77 - </br>
78 -
79 - ##### _RStudio_ has built-in features that are well suited to GIS!_
80 -
81 - ## Native graphic support for "maps"
82 -
83 - Charts, plots, images...
84 -
85 - ![(./images/rstudio_plot_pane.png)]

```

The console output shows the execution of the following R code:

```

> # TEST
> plot.new()
> plot(st_as_sf(wa_bbox, border='transparent'))
> plot(wa_pop, legend=TRUE, axes=FALSE, add=TRUE, legend.args = list(text = 'Population'))
> plot(wa$geometry, border='dark gray', add=TRUE)
> plot(st_as_sf(wa_bbox, border='black', add=TRUE))
> plot(wa_rural_towns$geometry, pch=20, cex=.5, add=TRUE)
> # TEST
> plot.new()
> plot(st_as_sf(wa_bbox, border='transparent'))
> plot(wa_pop, legend=TRUE, axes=FALSE, add=TRUE, legend.args = list(text = 'Population'))
> plot(wa$geometry, border='dark gray', add=TRUE)
> plot(st_as_sf(wa_bbox, border='black', add=TRUE))
> plot(wa_rural_towns$geometry, pch=20, cex=.5, col = 'dark gray', add=TRUE)
> plot(wa_rural_towns$geometry, pch=20, cex=.5, col='#5f6266', add=TRUE)
> plot(hosp_service_area,
+       border='red',
+       col='transparent',
+       add=TRUE)
> plot(candidate_towns$geometry,
+       add = TRUE,
+       col='transparent',
+       border = "darkblue")

```

The Environment pane shows the following data objects:

Object	Size	Variables
candidate_towns	13 obs.	10 variables
esri_places	31615 obs.	11 variables
hosp	50 obs.	11 variables
hosp_3857	7596 obs.	32 variables

The console also shows the execution of the following R code:

```

55
56
57 # Get the BBOX of all features
58 wa_bbox = st_bbox(wa)
59 wa_bbox
60 #   xmin      ymin      xmax      ymax
61 # -124.72284  45.54432 -116.91599  49.80249
62
63 # We want to pad out the S,W and E directions an extra degree
64 wa_bbox[2] = (wa_bbox[2] - 1.0) # S
65 wa_bbox[3] = (wa_bbox[3] + 1.0) # E
66 wa_bbox[1] = (wa_bbox[1] + 1.0) # W
67
68 wa_bbox
69 #   xmin      ymin      xmax      ymax
70 # -125.72284  44.54432 -115.91599  49.80249
71
72 # Plot the bounding box
73 plot(st_as_sf(wa_bbox))
74 plot(wa$geometry, add=TRUE)
75
76 # Final bounding box
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

# Built-in Help doc viewer

?raster

The screenshot displays the RStudio interface with a script editor on the left and the help viewer on the right. The script editor shows R code for creating a RasterLayer object from a bounding box and plotting it. The help viewer shows the documentation for the `raster` function, including its description, usage, and various methods.

```

54 # 1 53 01779804 040000U553 53 WA Washington 00 172112580220 12559278850 MULTIPOLYGON (((-123.2371 4...
55
56
57 # Get the BBOX of WA feature
58 wa_bbox <- st_bbox(wa)
59 wa_bbox
60 # xmin ymin xmax ymax
61 # -124.72584 45.54432 -116.91599 49.00249
62
63 # We want to pad out the S,W and E directions an extra degree
64 wa_bbox[1] <- (wa_bbox[1] - 1.0) # W
65 wa_bbox[2] <- (wa_bbox[2] - 1.0) # S
66 wa_bbox[3] <- (wa_bbox[3] + 1.0) # E
67
68 wa_bbox
69 # xmin ymin xmax ymax
70 # -125.72584 44.54432 -115.91599 49.00249
71
72 # Plot the bounding box
73 plot(st_as_sfc(wa_bbox))
74 plot(wa$geometry, add=TRUE)
75
76 gdalbuildvrt(gdalfile = "Data/gpw_v4_population_count_rev11_2020_30_sec.tif",
77 output.vrt = "Data/wa_gridded_pop.vrt",
78 te = wa_bbox)
79
80 # Let's see what this looks like
81 wa_pop <- raster("Data/wa_gridded_pop.vrt")
82 plot(st_as_sfc(wa_bbox))
83 plot(wa_pop, legend=TRUE, axes=FALSE, add=TRUE, legend.args = list(text = 'Population'))
84 plot(wa$geometry, add = TRUE)
85
86 # What does it look like if I only keep pixels > 0 and < 55
87
91:1 (Top Level)

```

The help viewer shows the following content:

## Create a RasterLayer object

### Description

Methods to create a RasterLayer object. RasterLayer objects can be created from scratch, a file, an Extent object, a matrix, an 'image' object, or from a Raster\*, Spatial\*, im (spatstat) asc, kasc (adehabitat\*), grf (geoR) or kde object.

In many cases, e.g. when a RasterLayer is created from a file, it does (initially) not contain any cell (pixel) values in (RAM) memory, it only has the parameters that describe the RasterLayer. You can access cell-values with [getValues](#), [extract](#) and related functions. You can assign new values with [setValues](#) and with [replacement](#).

For an overview of the functions in the raster package have a look here: [raster-package](#).

### Usage

```

## S4 method for signature 'character'
raster(x, band=1, ...)

## S4 method for signature 'RasterLayer'
raster(x)

## S4 method for signature 'RasterStack'
raster(x, layer=0)

## S4 method for signature 'RasterBrick'
raster(x, layer=0)

## S4 method for signature 'missing'
raster(nrows=180, ncols=360, xmin=-180, xmax=180, ymn=-90, ymx=90,
      crs, ext, resolution, vals=NULL)

## S4 method for signature 'Extent'
raster(x, nrows=10, ncols=10, crs="", ...)

## S4 method for signature 'matrix'

```

The console shows the execution of the script, including the output of the `?raster` command:

```

R 4.1.0 ~/CUGOS_2023_Spring_Fling/
> # Let's see what this looks like
> wa_pop <- raster("Data/wa_gridded_pop.vrt")
> plot(st_as_sfc(wa_bbox))
> plot(wa_pop, legend=TRUE, axes=FALSE, add=TRUE, legend.args = list(text = 'Population'))
> plot(wa$geometry, add = TRUE)
> # Let's see what this looks like
> wa_pop <- raster("Data/wa_gridded_pop.vrt")
> plot(st_as_sfc(wa_bbox))
> plot(wa_pop, legend=TRUE, axes=FALSE, add=TRUE, legend.args = list(text = 'Population'))
> plot(wa$geometry, add = TRUE)
starting httpd help server ... done
> ?raster|

```

# R itself has good geospatial support

- `raster` (which is now superseded by `terra`)

The raster package provides classes and functions to manipulate geographic (spatial) data in 'raster' format.

- `sf`

Support for simple features, a standardized way to encode spatial vector data. Binds to 'GDAL' for reading and writing data, to 'GEOS' for geometrical operations, and to 'PROJ' for projection conversions and datum transformations.

# Basic shapefile operations

- Open and read a shapefile

```
1 library(sf)
2 states <- st_read("Data/cb_2018_us_state_20m.shp")
3
4 Reading layer `cb_2018_us_state_20m' from data source
5   `/home/rstudio/CUGOS_2023_Spring_Fling/Data/cb_2018_us_state_20m.shp' using driver
6   Simple feature collection with 52 features and 9 fields
7   Geometry type: MULTIPOLYGON
8   Dimension:      XY
9   Bounding box:  xmin: -179.1743 ymin: 17.91377 xmax: 179.7739 ymax: 71.35256
10  Geodetic CRS:   4269
```

# Basic shapefile operations (cont.)

- Reproject to new CRS

```
1 states <- st_transform(states, crs = 4326)
2 states
3
4 Simple feature collection with 52 features and 9 fields
5 Geometry type: MULTIPOLYGON
6 Dimension: XY
7 Bounding box: xmin: -179.1743 ymin: 17.91377 xmax: 179.7739 ymax: 71.35256
8 Geodetic CRS: EPSG:4326
```

# Basic shapefile operations (cont.)

- Buffer around a set of points

```
1 wa_town_albers
2
3 Simple feature collection with 124 features and 9 fields
4 Geometry type: POINT
5 Dimension: XY
6 Bounding box: xmin: -2134539 ymin: 2773350 xmax: -1566342 ymax: 3155027
7 Projected CRS: EPSG:5070
8
9 # 30 miles approx 48,300 meters
10 wa_town_albers_buffer <- st_buffer(wa_town_albers, 48300)
11
12 wa_town_albers_buffer
13 Simple feature collection with 124 features and 9 fields
14 Geometry type: POLYGON
15 Dimension: XY
16 Bounding box: xmin: -2182839 ymin: 2725050 xmax: -1518042 ymax: 3203327
17 Projected CRS: EPSG:5070
```

# Basic raster operations

- Open a geotiff and display its extents

```
1 library(raster)
2 world_pop <- raster("Data/gpw_v4_population_count_rev11_2020_30_sec.tif")
3 world_pop
4
5 class      : RasterLayer
6 dimensions : 21600, 43200, 933120000 (nrow, ncol, ncell)
7 resolution : 0.008333333, 0.008333333 (x, y)
8 extent     : -180, 180, -90, 90 (xmin, xmax, ymin, ymax)
9 crs       : +proj=longlat +datum=WGS84 +no_defs
10 source    : gpw_v4_population_count_rev11_2020_30_sec.tif
11 names     : gpw_v4_population_count_rev11_2020_30_sec
```



# Basic raster operations (cont.)

- Create a VRT from larger raster using bbox as extents

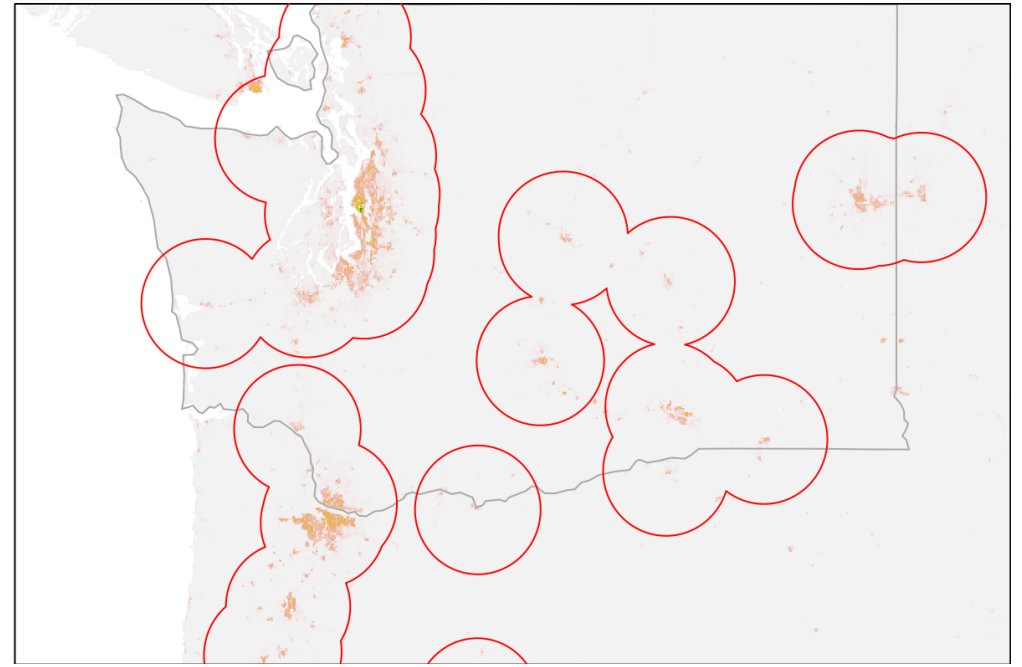
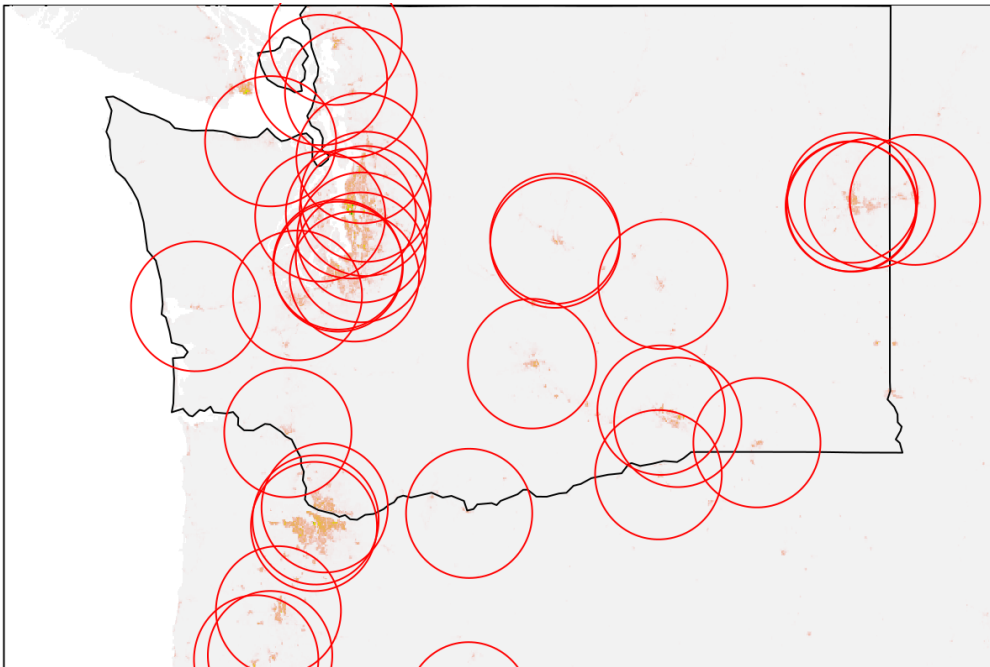
```
1 library(gdalUtilities)
2 gdalbuildvrt(gdalfile = "Data/gpw_v4_population_count_rev11_2020_30_sec.tif"
3               output.vrt = "Data/wa_gridded_pop.vrt",
4               te = wa_bbox)
```

- Load the VRT as a raster

```
1 wa_pop <- raster("Data/wa_gridded_pop.vrt")
2
3 setMinMax(wa_pop)
4 class      : RasterLayer
5 dimensions : 535, 1177, 629695 (nrow, ncol, ncell)
6 resolution : 0.008333333, 0.008333333 (x, y)
7 extent     : -125.7258, -115.9175, 44.54416, 49.00249 (xmin, xmax, ymin, ymax)
8 crs       : +proj=longlat +datum=WGS84 +no_defs
9 source    : wa_gridded_pop.vrt
10 names    : wa_gridded_pop
11 values   : 0, 7647.292 (min, max)
```

# Dissolve operation

```
1 # Add 'type' field to dissolve on
2 hosp_buffer$type <- "combined_service_areas"
3
4 hosp_service_area <- hosp_buffer %>%
5   group_by(type) %>%
6   summarise()
```



# A practical example

Assume that you work for an agency that is responsible for helping to expand access to healthcare in rural America. You have been tasked to come up with a list of 3 rural cities, or towns, in the State of Washington that would most benefit from the construction of a new hospital.

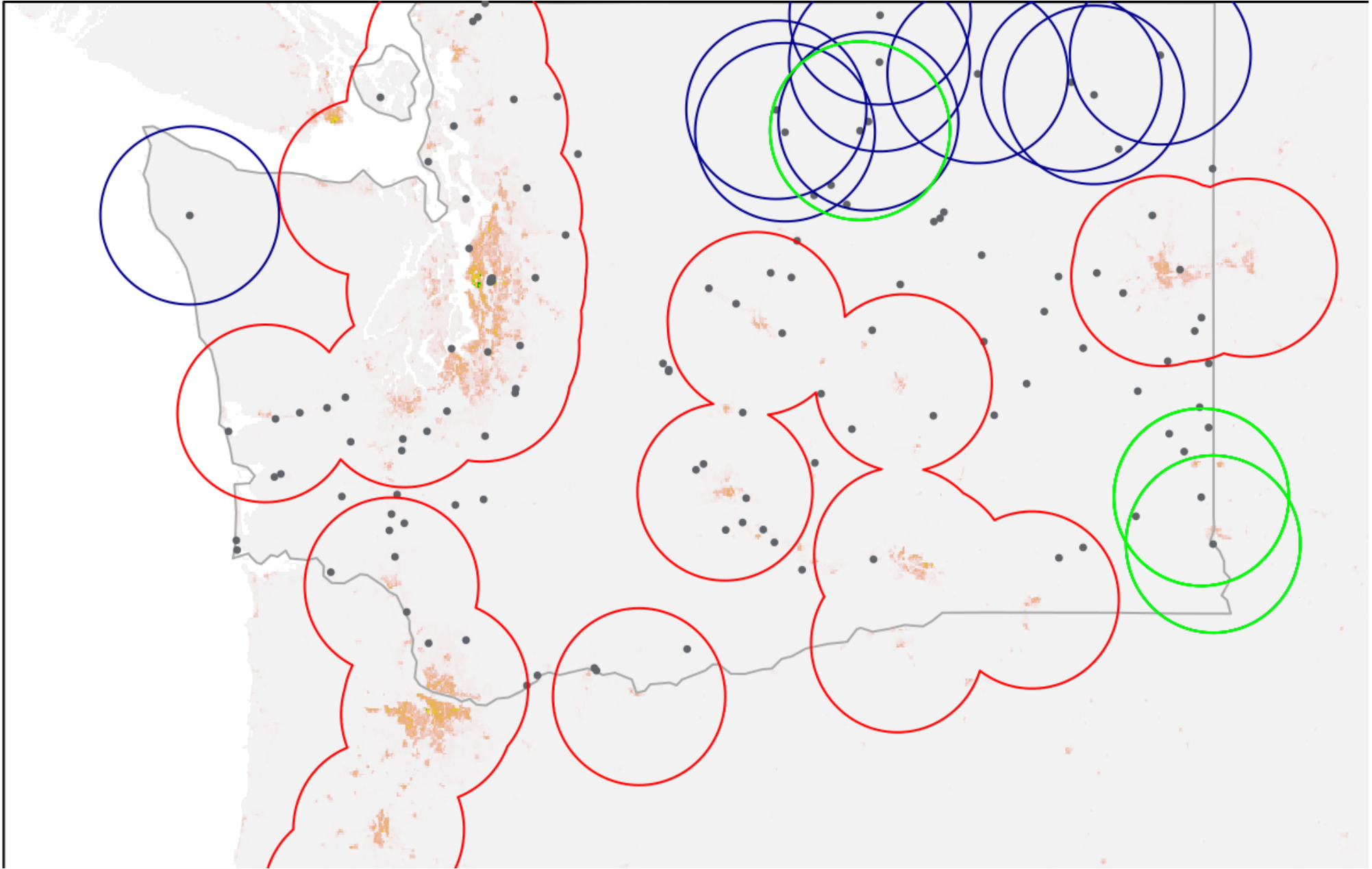
Provide the names for the top 3 cities or towns and the estimated population that would be covered by a 30 mile service area around each of them.

# Combining rasters and vectors

Where this gets fun!

```
1 st_as_sf(raster::extract(wa_pop,
2   candidate_towns,
3   fun=sum,
4   na.rm=TRUE,
5   sp=TRUE)) -> candidate_town_pops
6
7 st_drop_geometry(candidate_town_pops) %>%
8   select(NAME, wa_gridded_pop) %>%
9   arrange(desc(wa_gridded_pop)) %>%
10  slice(1:3)
11
12      NAME wa_gridded_pop
13 1   Colton   144817.03
14 2   Asotin   133227.72
15 3 Okanogan    36177.21
```

# Final results - Top 3 candidates



# Additional inspiration

- Shiny apps that showcase interactivity with maps

<https://fitzlab.shinyapps.io/cityapp/>

<https://alexh5.shinyapps.io/INFO201FinalProject/>

[https://vac-lshtm.shinyapps.io/ncov\\_tracker/](https://vac-lshtm.shinyapps.io/ncov_tracker/)