Google Earth Engine guide: Part 2

- Previously, we extracted raster data from the earth engine code editor.
- Now, we are going to now see how to do this directly in R using the `rgee` package. For additional reference, please see https://csaybar.github.io/rgee-examples/.
- Note that this process requires an active version of python on your computer and the installation process can be somewhat involved, especially for windows OS. RGEE Installation

---

```
#ee_install() # only need to do once
#ee_Authenticate() # need to do once a week
ee_Initialize()
```

```
-- rgee 1.1.7 --------------------------------------- earthengine-api 0.1.370 --
 v user: not_defined
 v Initializing Google Earth Engine:
 v Initializing Google Earth Engine:  DONE!

 v Earth Engine account: users/andyhoegh

 v Python Path: /Users/andyhoegh/.virtualenvs/rgee/bin/python
--------------------------------------------------------------------------------
```

```
ee_check()
```

```
(*)  Python version
v [Ok] /Users/andyhoegh/.virtualenvs/rgee/bin/python v3.12
(*)  Python packages:
v [Ok] numpy
v [Ok] earthengine-api
```

As an example, consider digital elevation from the HydroSHEDS data

First, for comparison with last time we can run this JS code directly in earth engine editor.

```
var dataset = ee.Image('WWF/HydroSHEDS/03CONDEM');
var elevation = dataset.select('b1');
var elevationVis = {
  min: -50.0,
  max: 3000.0,
  gamma: 2.0,
};
Map.setCenter(-111.05, 45.667, 11);
Map.addLayer(elevation, elevationVis, 'Elevation');
```

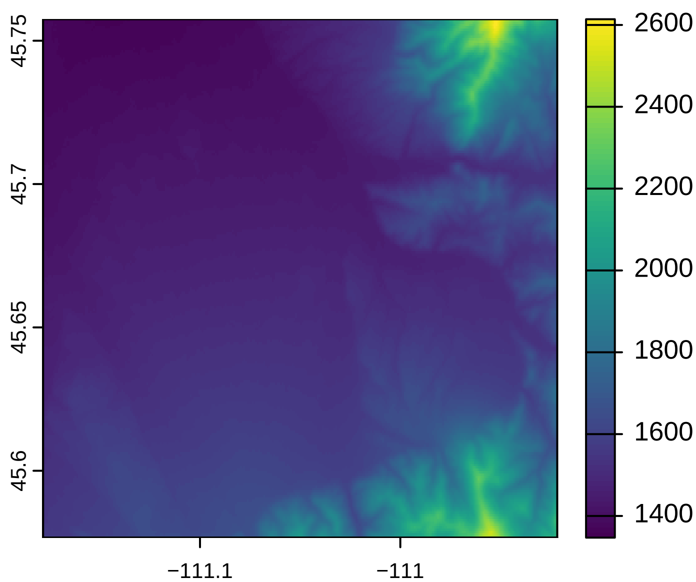
Here is the `rgee` analog that also extracts data for a 10KM buffer around MSU.

```r
elevation <- ee$Image("WWF/HydroSHEDS/03CONDEM")
bozeman <- ee$Geometry$Point(-111.05,45.667)$buffer(10000)$bounds()

boz_elev_raster <- ee_as_rast(elevation, bozeman, via = 'drive')
```

```
NOTE: Google Drive credentials were not loaded. Running ee_Initialize(user = 'ndef', drive =

Registered S3 method overwritten by 'geojsonsf':
  method        from
  print.geojson geojson

- region parameters
 sfg      : POLYGON ((-111.1777 45.5770 .... .75699, -111.1777 45.57705))
 CRS      : GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563, .....
 geodesic : FALSE
 evenOdd  : TRUE

- download parameters (Google Drive)
 Image ID     : 03CONDEM
 Google user : ndef
```

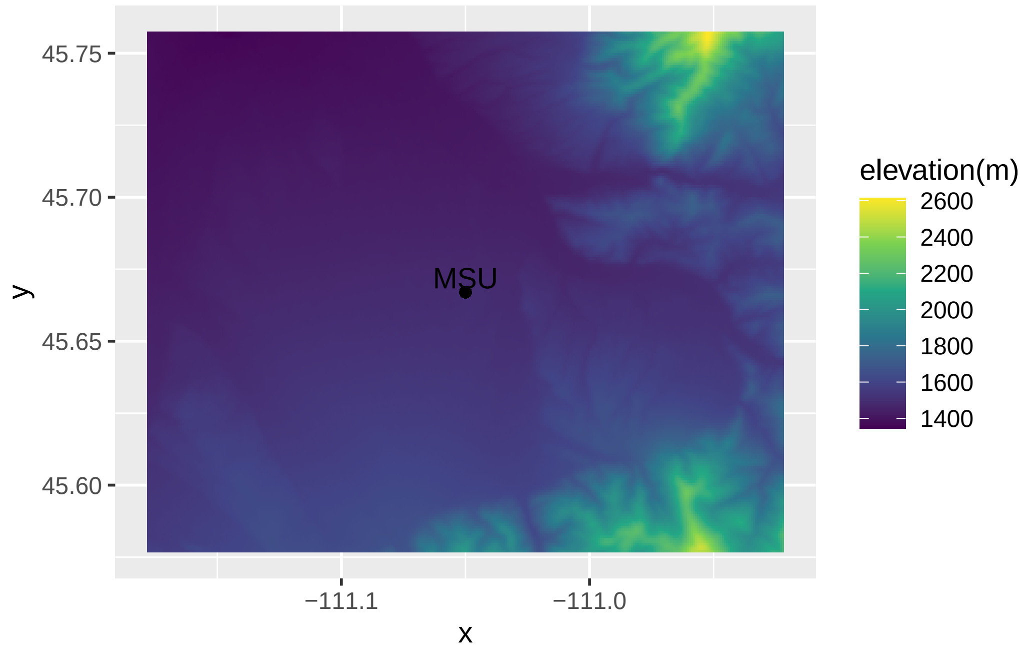
```
 Folder name : rgee_backup
 Date        : 2025_02_12_13_39_02
Polling for task <id: XSZOCU6UCJUA52Y4LN36ZABQ, time: 0s>.
Polling for task <id: XSZOCU6UCJUA52Y4LN36ZABQ, time: 5s>.
Polling for task <id: XSZOCU6UCJUA52Y4LN36ZABQ, time: 10s>.
Polling for task <id: XSZOCU6UCJUA52Y4LN36ZABQ, time: 15s>.
State: COMPLETED
Moving image from Google Drive to Local ... Please wait
```

```
plot(boz_elev_raster)
```



Recall, we can even create a data frame with the raster information and use this in tidyverse.

```
boz_df <- as.data.frame(boz_elev_raster, xy = T)
boz_df |>
  mutate(`elevation(m)` = b1) |>
  ggplot() +
    geom_raster(aes(x = x, y = y, fill = `elevation(m)`)) +
    scale_fill_viridis_c() +
  geom_point(x = -111.05,y = 45.667) +
  annotate('text', label = 'MSU', x = -111.05,y = 45.672)
```



Putting it all together

Recall the elk dataset from HW1

Step 1: Data Visualization

```
elk <- read_csv('https://raw.githubusercontent.com/Stat534/data/refs/heads/main/elk.csv')
```

```
Rows: 5924 Columns: 13
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (6): comments, sensor-type, individual-taxon-canonical-name, tag-local-...
dbl  (5): event-id, location-long, location-lat, migration-stage, study-spec...
lgl  (1): visible
time (1): timestamp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

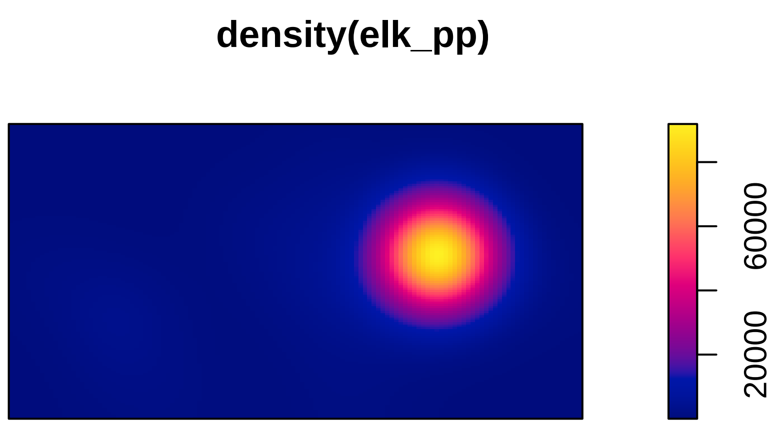
Step 2: Is this a homogenous PP?

```
elk_pp <- ppp(y = elk$`location-lat`, x = elk$`location-long`,
              window = owin(yrange = c(min(elk$`location-lat`), max(elk$`location-lat`)),
                   xrange = c(min(elk$`location-long`), max(elk$`location-long`))))
```

```
Warning: data contain duplicated points
```

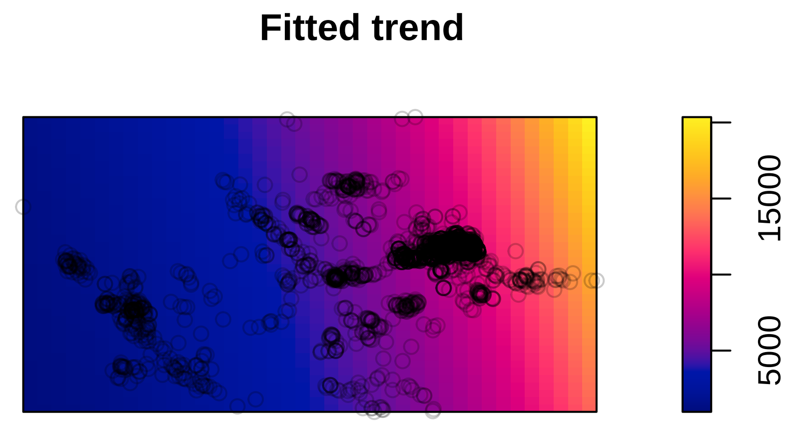
Step 3: Intensity Surface

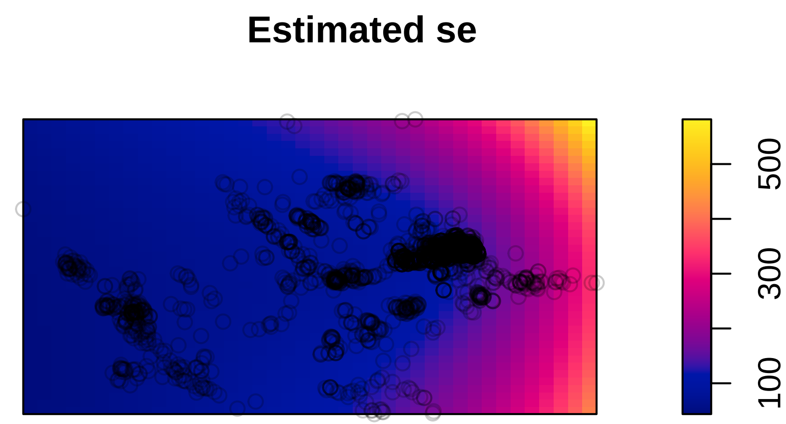
```
plot(density(elk_pp))
```



There is not an obvious parametric intensity function of Lat / long. So let's start with a naive (log) linear specification - which unsurprisingly results in a poor fit.

```
naive_ppm <- ppm(elk_pp ~ x + y)
plot(naive_ppm)
```





Step 4: Geospatial Covariates

There is likely more to the story, so let's pull elevation from GEE, but we need to make sure the bounding box matches our ppm. See this for bounding box help.

```
elk_box <- ee$Geometry$BBox(min(elk$`location-long`),
                            min(elk$`location-lat`),
                            max(elk$`location-long`),
                            max(elk$`location-lat`))
```

You might need this function to convert the SpatRaster to an im object

```
#https://stackoverflow.com/questions/77912041/convert-raster-terra-to-im-object-spatstat
as.im.SpatRaster1 <- function(X) {
    X <- X[[1]]
    rs <- terra::res(X)
    e <- as.vector(terra::ext(X))
    out <- list(
        v = as.matrix(X, wide=TRUE)[nrow(X):1, ],
        dim = dim(X)[1:2],
        xrange = e[1:2],
        yrange = e[3:4],
        xstep = rs[1],
        ystep = rs[2],
        xcol = e[1] + (1:ncol(X)) * rs[1] + 0.5 * rs[1],
        yrow = e[4] - (nrow(X):1) * rs[2] + 0.5 * rs[2],
        type = "real",
        units  = list(singular=units(X), plural=units(X), multiplier=1)
    )
    attr(out$units, "class") <- "unitname"
    attr(out, "class") <- "im"
    out
}
```

Step 5: Diagnostics & Model Choice

As with general statistical modeling frameworks, we can visualize model fit & residuals (`diagnose.ppm`). These models also have a built in likelihood, so you can also use `AIC`