Geo_Data_Sci_R

Wyclife Agumba Oluoch

1/22/2021

Loading Libraries to use in this project

Loading the necessary packages to use in the codes which will be helpful in analyzing geographic data in R. font_import() helps to make the fonts brought by `extrafont` available to R for use.

The data I used work can be directly downloaded, extracted from zipped folder, and saved in the sub-folder called *shp* inside the .Rmd file folder.

The next step is to read in the data using `st_read()`. I will call the created object honeybee

```
## Reading layer ‘HoneyBeePermits2017’ from data source ‘D:\R\SPATIAL\spatialwithR\shp\HoneyBeePermits20
## Simple feature collection with 90 features and 3 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -93.32817 ymin: 44.89744 xmax: -93.20644 ymax: 45.03776
## geographic CRS: WGS 84
```

There is a lot of information which we get by loading the data including path to the data-set being loaded, driver used to load the file, geometry type, dimension, bounding box, and CRS. We can get information about the sf object just like with normal data-frame or tibble, for example we can simply check the structure by:

```
str(honeybee)
```

```
## Classes ’sf’ and ’data.frame’:   90 obs. of  4 variables:
##  $ FID     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Address : chr  "1023 MOUNT CURVE AVE" "1108 25TH AVE N" "1235 WASHBURN AVE N" "1300 NICOLLET AVE"
##  $ HiveType: chr  "GROUND" "GROUND" "GROUND" "ROOFTOP" ...
##  $ geometry:sfc_POINT of length 90; first list element:  ’XY’ num  -93.3 45
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA
##   ..- attr(*, "names")= chr [1:3] "FID" "Address" "HiveType"
```

This is telling us that we are working with a point feature. Such a cool information to have. Now we may need to know the coordinate reference system of our data. This is a very important step before running any analyses on the data as it has significant effect on the units of measurements, for example length, area, among others. Good to know that most of the functions on sf objects start with **st_**. This will be evident as we progress with the project. Here we start with `st_crs()`.

```
st_crs(honeybee)
```

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

That is returning quite helpful information. We can also have the information within the text as WGS 84, GEOGCRS[“WGS 84”, DATUM[“World Geodetic System 1984”, ELLIPSOID[“WGS 84”,6378137,298.257223563, LENGTHUNIT[“metre”,1]]], PRIMEM[“Greenwich”,0, ANGLEUNIT[“degree”,0.0174532925199 CS[ellipsoidal,2], AXIS[“latitude”,north, ORDER[1], ANGLEUNIT[“degree”,0.0174532925199433]], AXIS[“longitude”,east, ORDER[2], ANGLEUNIT[“degree”,0.0174532925199433]], ID[“EPSG”,4326]]. I still can’t imagine how the output will look like in the text. Let us see by knitting the markdown. Everything thrown to text :).

Now the next step would be to check for self intersection in our data.

```
sum(st_is_valid(honeybee) == TRUE)
```

```
## [1] 90
```

We do not have any problem of self-intersection in our data-set. All the 90 points are considered valid. Behind the scene, there are 90 TRUEs and each TRUE is equated to 1 hence the sum being 90. Okay this is reminding me of something I was working on and worried how to solve it in QGIS without much success. That is I accidentally criss-crossed same line feature while digitizing rivers and I could not trace the individual river to edit it. I will create dummy feature with such error in QGIS and load it to check its validity.

```
dummy <- st_read('shp/dummy.shp')
```

```
## Reading layer ‘dummy’ from data source ‘D:\R\SPATIAL\spatialwithR\shp\dummy.shp’ using driver ‘ESRI
## Simple feature collection with 2 features and 1 field
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 35.40398 ymin: 3.36293 xmax: 35.49257 ymax: 3.399832
## geographic CRS: WGS 84
```

The dummy object is read seamlessly and having 2 features. In other words, I have two rivers. It is of geometry type LINESTRING. I know one of the two lines has crossed itself at some point. Let me now check for its validity.

```
sum(st_is_valid(dummy) == TRUE)
```

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

This returned TRUEs even though I know pretty sure that one of the lines has cut itself at some point(s). Let me try this with polygon features. I will create three polygons, one kind of a rectangle and the other two having 'funny' shapes which cut themselves. I think that the latter two should be invalid polygon. Here we go.

```
dummy_poly <- st_read('shp/dummy_poly.shp')
```

```
## Reading layer 'dummy_poly' from data source 'D:\R\SPATIAL\spatialwithR\shp\dummy_poly.shp' using dri
## Simple feature collection with 3 features and 1 field
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 35.03253 ymin: 3.402807 xmax: 35.64507 ymax: 3.648782
## geographic CRS: WGS 84
```

Now let me check for the validity of the features I have in my data-set of the dummy_poly. Fingers crossed. I need errors :) seriously.

```
sum(st_is_valid(dummy_poly) == TRUE)
```

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

Good, the point is now well sank home. Only one polygon is valid. The other two are intersecting with themselves hence considered invalid as polygon features.

Before I proceed to honeybee work, I want to create a point sf with some points having exactly the same coordinates and see whether such are considered invalid. This can help me solve the problem of duplicates of occurrence records in species distribution modeling later.

```
points_sf <- data.frame(lon = c(35.017893, 35.017893,35.010720, 34.587676),
                        lat = c(-0.231059, -0.231059, -0.237188, -0.412132)) %>%
  st_as_sf(coords = c('lon', 'lat')) %>%
  st_set_crs(4326)
```

Then I check for validity of the sf object which I know has four points with one being duplicated.

```
sum(st_is_valid(points_sf) == TRUE)
```

```
## [1] 4
```

Okay, from this I conclude that this validity is not applicable in case of duplicated points as all my four points have been regarded valid. So, I may be right to say that duplicate points are not intersecting features, maybe.

Now back to honeybee work. I want to project it to UTM zone 15N for Minnesota. I remember that earlier, while working with geocomputation with R, I developed (with the help of Geocomputation with R book mentioned earlier) a function which can help with knowing the EPSG code for any place on earth provided you know long and lat of the place, which I can get from Google Earth search for Minnesota.

```
lonlat2UTM <- function(lonlat){
  utm <- (floor((lonlat[1] + 180)/6) %% 60) + 1
  if(lonlat[2] > 0){
    utm + 32600
  } else{
    utm + 32700
  }
}
```

Cool, the function is created, now I can call it to get the epsg code best suited for Minnesota (long = -94.685798°, lat = 46.729741°).

```
lonlat2UTM(c(-94.685798, 46.729741))
```

```
## [1] 32615
```

This is returning 3.2615×10^4 which is different from 26915 which has been used in the tutorial. Okay, they are using different datums or data :). 26915 is for NAD and 32615 is for WGS84. The former is preferred in this case as it represents North America best.

Now I can project the honeybee data to my desired EPSG code using st_transform function. If this was in **Python** with `GeopandasGeoDataFrame` then I would use honeybee.to_crs('EPSG:26915'). The R one looks fine but the **Python** one seems easier to understand what is going on. Probably why Python is considered simpler than R. No discussion or argument please, stay with your views.

```
honeybee_utm <- st_transform(honeybee, 26915)
```

We can check the crs of honeybee_utm.

```
## Coordinate Reference System:
##   User input: EPSG:26915
##   wkt:
## PROJCRS["NAD83 / UTM zone 15N",
##     BASEGEOGCRS["NAD83",
##         DATUM["North American Datum 1983",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4269]],
##     CONVERSION["UTM zone 15N",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",-93,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["North America - 96Â°W to 90Â°W and NAD83 by country"],
##         BBOX[25.61,-96,84,-90]],
##     ID["EPSG",26915]]
```

Quite clear that the projection has been done successfully. This is now kind of the best representation of the geographic properties of the place (direction, area, and distance).

Now to plot these points, we may need to have a background map. Otherwise just points on a white background will show scatter plot kind which is not that mappy or map-like. So I will import some free background map and assign it same projection as the honeybee points.

```
neighborhoods <- st_read("https://opendata.arcgis.com/datasets/055ca54e5fcc47329f081c9ef51d038e_0.geojs
  st_transform(26915)
```

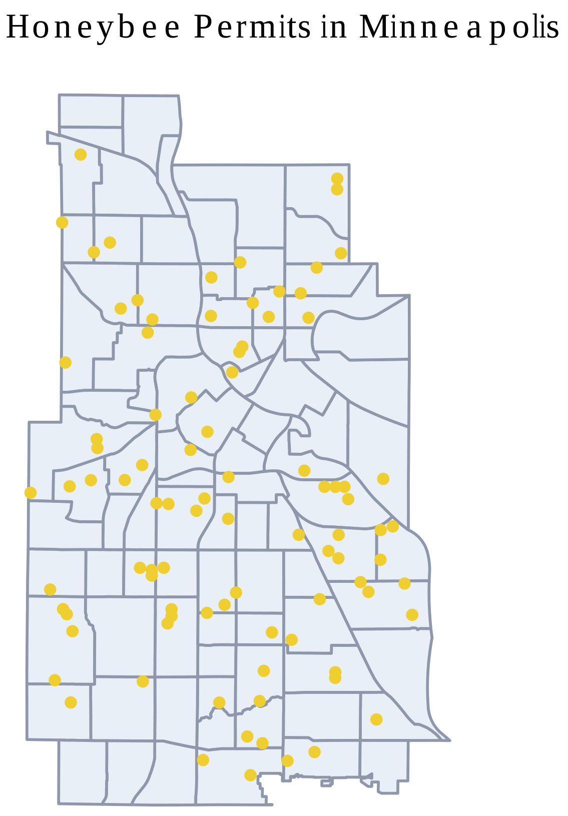
```
## Reading layer `Minneapolis_Neighborhoods' from data source `https://opendata.arcgis.com/datasets/055
## Simple feature collection with 87 features and 10 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -93.32911 ymin: 44.89059 xmax: -93.19433 ymax: 45.05125
## geographic CRS: WGS 84
```

Cool, the original link (which is shown on the original source) failed and I had to navigate to the source to pick the new link which has then worked fine. Good.

I will load the necessary `extrafont` package as I prepare to use `ggplot2` for the plotting of the map and use some unique font in title label.

Now I can simply go ahead and generate the plot. Colors can be changed as need may be.

```
ggplot() +
  geom_sf(data = neighborhoods, fill = '#e8eff7', color = '#8f98aa') +
  geom_sf(data = honeybee_utm, color = '#efcf2f') +
  theme_minimal() +
  theme(panel.grid.major = element_line('transparent'),
        axis.text = element_blank(),
        text = element_text(family = "Century Gothic")) +
  ggtitle('Honeybee Permits in Minneapolis')
```



That is a pretty fine map. Though things like North arrow, scale, graticule, border are still missing.

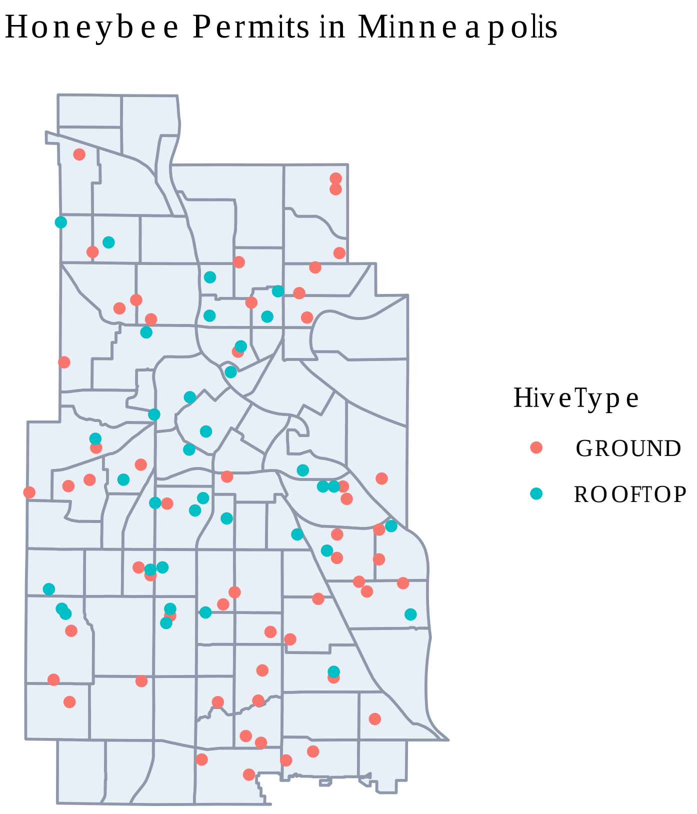
We can use the HiveType variable in the data-set to color the points so that we can visually appreciate the distribution of hives by type within the area. To remind myself of the names of the variables in the sf:

```
colnames(honeybee_utm)
```

```
## [1] "FID"      "Address"  "HiveType" "geometry"
```

To use the variable of HiveType to color the points we make slight modification to our earlier plot.

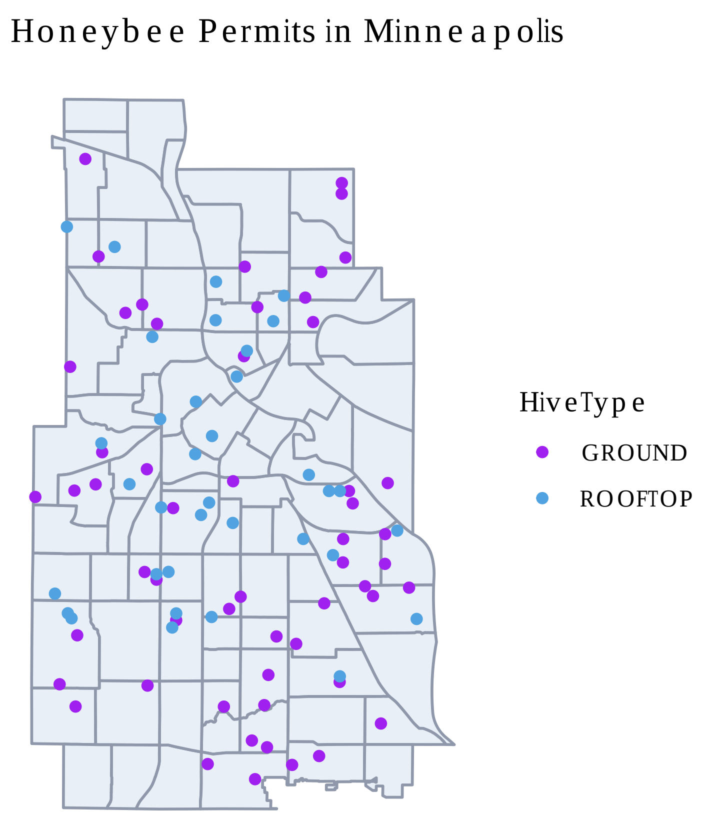
```
ggplot() +
  geom_sf(data = neighborhoods, fill = '#e8eff7', color = '#8f98aa') +
  geom_sf(data = honeybee_utm, aes(color = HiveType)) +
  theme_minimal() +
  theme(
    panel.grid.major = element_line('transparent'),
    axis.text = element_blank(),
    text = element_text(family = "Century Gothic")
  ) +
  ggtitle('Honeybee Permits in Minneapolis')
```



Great.Nice to see how the different types of hives are distributed across the Minneapolis region.

In case I am not happy with the colors of the points, I can assign them manually as:

```
ggplot() +
  geom_sf(data = neighborhoods, fill = '#e8eff7', color = '#8f98aa') +
  geom_sf(data = honeybee_utm, aes(color = HiveType)) +
  scale_color_manual(values = c('purple', '#50a2e0')) +
  theme_minimal() +
  theme(
    panel.grid.major = element_line('transparent'),
    axis.text = element_blank(),
    text = element_text(family = "Century Gothic")
  ) +
  ggtitle('Honeybee Permits in Minneapolis')
```



Now I am interested in knowing the number of honeybee permits per neighborhood That is, how many points fall within each of the polygons. This is calling for spatial joins concept.

Spatial joins

The type of join to use here is left join which is the default. Loosely speaking, I will say.

```
## Simple feature collection with 6 features and 13 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 474840.8 ymin: 4977247 xmax: 480684.2 ymax: 4988693
## projected CRS:  NAD83 / UTM zone 15N
##   FID.x            BDNAME BDNUM TEXT_NBR Shape_STAr Shape_STLe
## 1     1      Phillips West    90       90   10669250   14403.89
## 2     2      Downtown West    87       87   20756130   19220.60
## 3     3      Downtown East    88       88   10254989   13436.60
## 4     4    Ventura Village    89       89   12635259   16988.53
## 5     5 Sumner - Glenwood     29       29    5741860   11065.34
## 6     6      Shingle Creek     1       01   13900424   15768.00
##                                              NCR_LINK IMAGE SHAPE_Length
## 1 http://www.nrp.org/r2/Neighborhoods/Orgs/PHW.html   PHW   0.04580106
## 2 http://www.nrp.org/r2/Neighborhoods/Orgs/DTN.html   DTN   0.06367142
## 3 http://www.nrp.org/r2/Neighborhoods/Orgs/DTN.html   DTN   0.04517872
## 4 http://www.nrp.org/r2/Neighborhoods/Orgs/VEN.html   VEN   0.05958976
## 5 http://www.nrp.org/r2/Neighborhoods/Orgs/SGL.html   SGL   0.03553546
## 6 http://www.nrp.org/r2/Neighborhoods/Orgs/SHC.html   SHC   0.05504762
##     SHAPE_Area FID.y           Address HiveType                       geometry
## 1 1.130216e-04    26   2600 PARK AVE S  ROOFTOP MULTIPOLYGON (((479290.6 49...
## 2 2.199649e-04    88 821 MARQUETTE AVE  ROOFTOP MULTIPOLYGON (((479493.6 49...
## 3 1.086748e-04    NA              <NA>     <NA> MULTIPOLYGON (((480684.2 49...
## 4 1.338686e-04    12   1823 PARK AVE S   GROUND MULTIPOLYGON (((480318 4979...
## 5 6.085638e-05    NA              <NA>     <NA> MULTIPOLYGON (((477273.4 49...
## 6 1.474880e-04    NA              <NA>     <NA> MULTIPOLYGON (((475553.4 49...
```

Now we can go ahead and use dplyr verbs to count per unit/

```
nb <- nb_join %>%
  group_by(BDNAME) %>%
  summarise(n_permits = n())
```

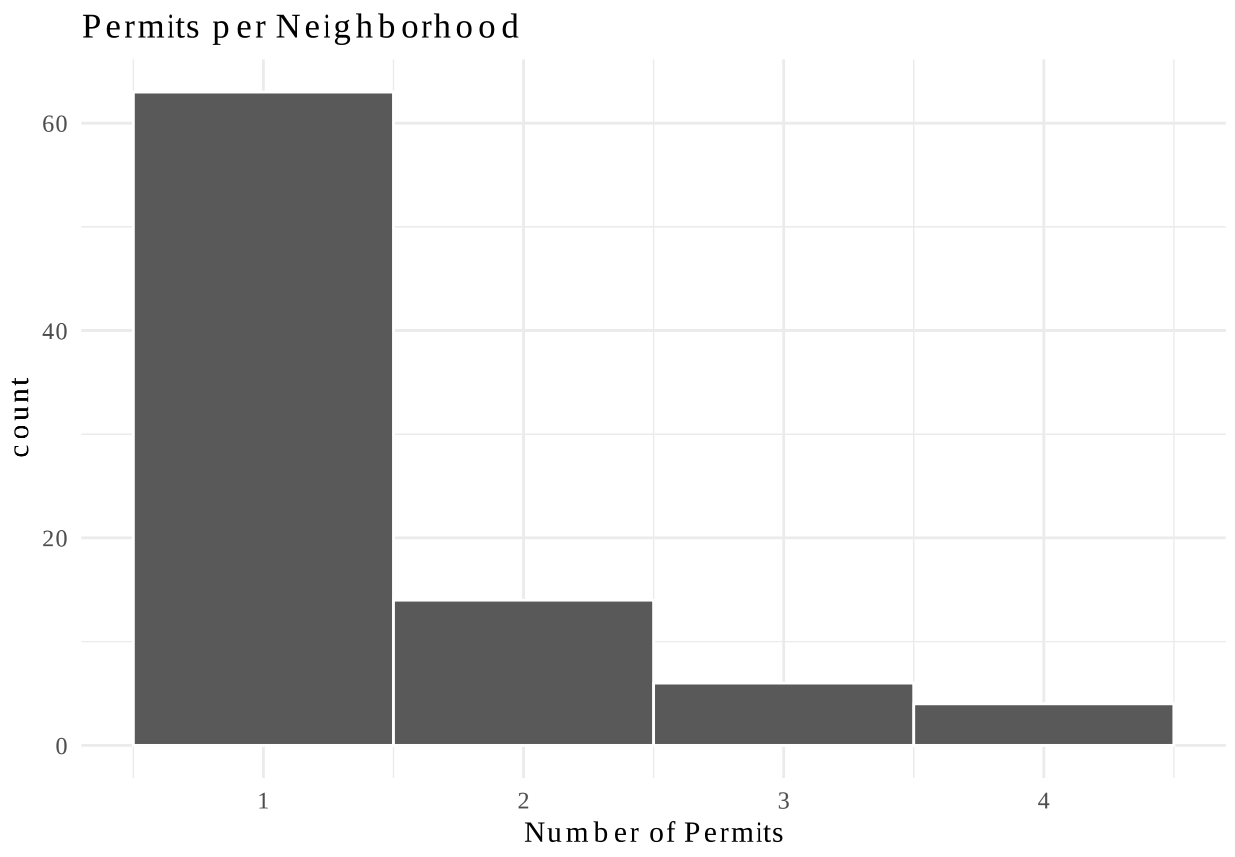
We can then arrange by the number of permits to see which has the highest number of honeybee permits and which has least.

```
nb %>%
  arrange(-n_permits)
```

```
## Simple feature collection with 87 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 474025 ymin: 4970837 xmax: 484658.8 ymax: 4988693
## projected CRS:  NAD83 / UTM zone 15N
## # A tibble: 87 x 3
##    BDNAME     n_permits                                             geometry
##    <chr>          <int>                                   <MULTIPOLYGON [m]>
##  1 Howe               4 (((484138.7 4976477, 484137 4976417, 484136.4 4976395, ~
##  2 Linden Hi~         4 (((474644.4 4976084, 474645 4976084, 474754.1 4976083, ~
##  3 Seward             4 (((480931.1 4979016, 480974 4979016, 481000.6 4979016, ~
##  4 Whittier           4 (((478742.6 4978841, 478742.6 4978839, 478742.1 4978766~
##  5 Bryn - Ma~         3 (((475203.9 4980858, 475207.1 4980858, 475207.2 4980858~
##  6 Cooper             3 (((483614.9 4977753, 483649.7 4977736, 483656.1 4977733~
##  7 King Field         3 (((478327.5 4976072, 478327.6 4976062, 478328 4975843, ~
##  8 Logan Park         3 (((480318 4983859, 480366.8 4983858, 480367.5 4983858, ~
##  9 Longfellow         3 (((481337.9 4977869, 481387.2 4977857, 481400.5 4977857~
## 10 South Upt~         3 (((477273.1 4977255, 477273.1 4977253, 477273.1 4977252~
## # ... with 77 more rows
```

We can have a quick look at the distribution in form of histogram.

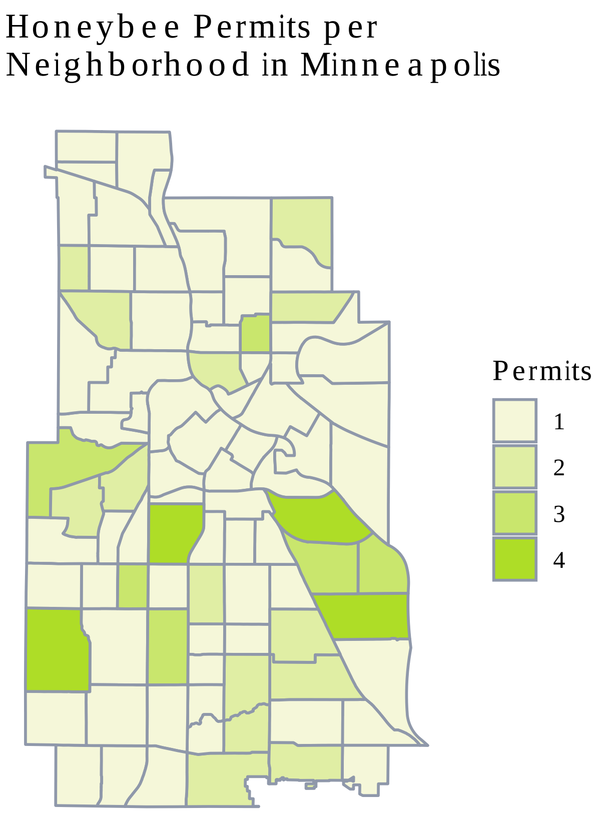
```
ggplot(nb, aes(x = n_permits)) +
  geom_histogram(binwidth = 1, color = 'white') +
  theme_minimal() +
  ggtitle('Permits per Neighborhood') +
  theme(text = element_text(family = 'Century Gothic')) +
  labs(x = 'Number of Permits')
```



Majority of the neighborhoods (over 60) had only one permit. The maximum number of permits per neighborhood was 4 which was in four neighborhoods (Howe, Linden Hills, Seward, ad Whittier).

It is possible to color the map by number of permits.

```r
ggplot(nb) +
  geom_sf(aes(fill = n_permits), color = '#8f98aa') +
  scale_fill_gradient(low = '#f5f7d9',
                      high = '#aedd27',
                      guide = guide_legend(title = 'Permits')) +
  theme_minimal() +
  theme(panel.grid.major = element_line('transparent'),
        axis.text = element_blank(),
        text = element_text(family = 'Century Gothic')) +
  ggtitle('Honeybee Permits per \nNeighborhood in Minneapolis')
```



Checking the possible existence of spatial autocorrelation in the data. That is, are the points existence related?

```
neighborhoods_sp <- as(nb, 'Spatial') # Converting the sf to sp
nb_obj <- poly2nb(neighborhoods_sp)   # Create neighborhood object
summary(nb_obj)
```

```
## Neighbour list object:
## Number of regions: 87
## Number of nonzero links: 492
## Percentage nonzero weights: 6.500198
## Average number of links: 5.655172
## Link number distribution:
##
##  2  3  4  5  6  7  8  9
##  2  6 15 18 19 14  9  4
## 2 least connected regions:
## 69 80 with 2 links
## 4 most connected regions:
## 40 61 76 78 with 9 links
```

```
weights <- nb2listw(nb_obj, style = 'B') # Creates a matrix of binary spatial weights (connected or not
```

```
moran(neighborhoods_sp$n_permits, weights, n = length(weights$neighbours), S0 = Szero(weights))
```

```
## $I
## [1] 0.08526573
##
## $K
## [1] 5.602669
```

The Moran's I statistic is 0.085, very slight positive autocorrelation. Is this significant or not? We will use Monte Carlo Simulation for this test. This will randomly assign values to the polygons and calculates Moran's I for each iteration. It then compares the observed statistic to the generated distribution.

```
set.seed(123)
moran.mc(neighborhoods_sp$n_permits, weights, nsim = 9999) # 10000 simulations
```

```
##
##  Monte-Carlo simulation of Moran I
##
## data:  neighborhoods_sp$n_permits
## weights: weights
## number of simulations + 1: 10000
##
## statistic = 0.085266, observed rank = 9325.5, p-value = 0.06745
## alternative hypothesis: greater
```

Being that our observed statistic is 0.085 with a p-value of 0.0675, we would say that our distribution is not different from a randomly distributed. Therefore, we now know that the honeybee permits are randomly distributed in Minneapolis at the neighborhood level. It's important to note this spatial unit, clustering calculations can be very different at different units. Meaning if it was conducted throughout the world then for sure some regions would be having more honeybees than other and show statistically significant autocorrelation. For example, the whole of Antarctica might be devoid of honeybee permits.