Spatial analysis and the sf package

Alys Young and Michael Traurig

May 2022

QAEco Coding Club

Code: https://github.com/AlysY/sf_package_tute

Alys Young

Michael Traurig

Spatial analysis in R

Packages:

- Rgdal - retired in 2023
- Raster and terra, sf and sp

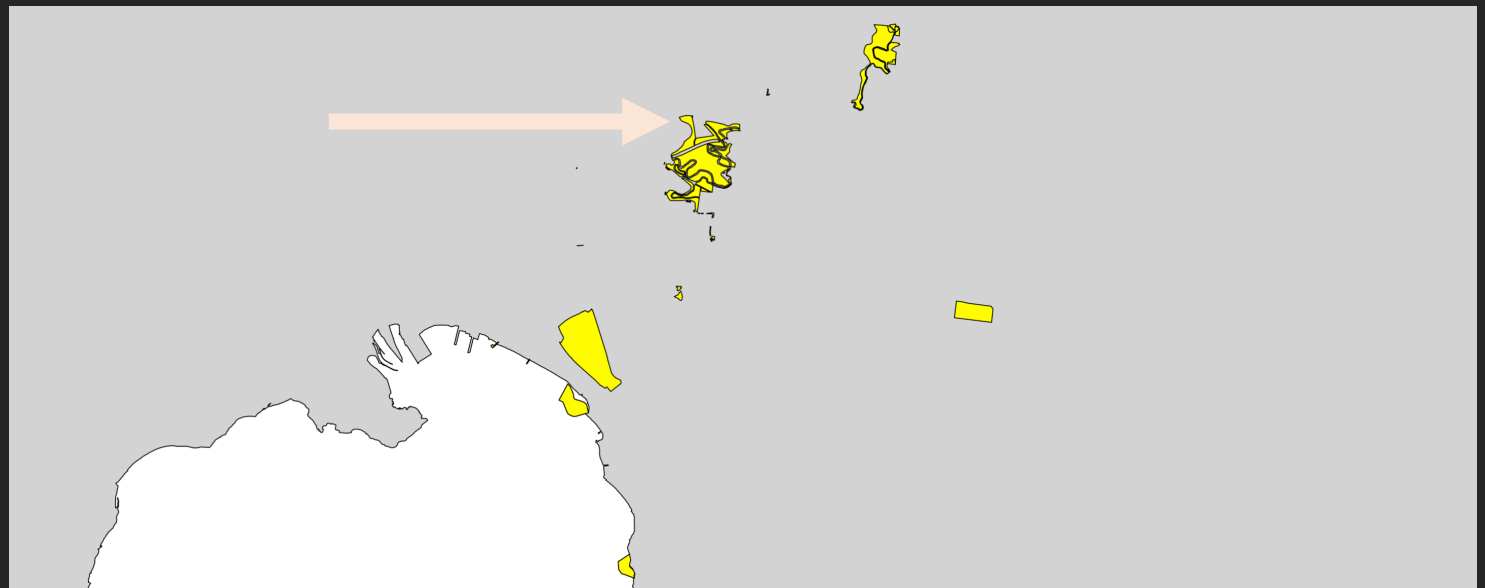
Data:

- Point, line, polygon data

Sf package:

- Dataframe structure with geometry column

```
> parks
Simple feature collection with 49 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 318561.4 ymin: 580 294 xmax: 333766 ymax: 5819044
Projected CRS: GDA2020 / MGA zone 55
First 10 features:
                                    NAME                                geom
1                        Yarra Bend Park MULTIPOLYGON (((324415.5 58...
2                  Lower Yarra River land MULTIPOLYGON (((325329.9 58...
3                  Lower Yarra River land MULTIPOLYGON (((325252.7 58...
4                  Lower Yarra River land MULTIPOLYGON (((325321.3 58...
5                        Yarra Bend Park MULTIPOLYGON (((326111 5815...
6             St Kilda Pier and Breakwater MULTIPOLYGON (((321511.8 58...
7                  Lower Yarra River land MULTIPOLYGON (((324537.1 58...
8                        Yarra Bend Park MULTIPOLYGON (((325839.1 58...
9               Part Yarra Valley Parklands MULTIPOLYGON (((330807 5817...
10 Parks Victoria Depot - Barkly Ave Park MULTIPOLYGON (((324240.5 58...
```



Spatial analysis in R

Packages:

- Rgdal - retired in 2023
- Raster and terra, sf and sp

Data:

- Point, line, polygon data

Sf package:

- Dataframe structure with geometry column
- Functions are st_***

Common functions – st_read, st_write and st_as_sf

st_read

- Read in files

st_write

- Write files

st_as_sf

- Convert a dataframe with the coordinates as a column, into a spatial object

In R:

```
# read in the data ------------------------------------------------------------
## New functions:
?st_read
?st_as_sf



## Melbourne parks - multipolygon
parks <- st_read("data/basic_intro/parks.gpkg")

head(parks)
summary(parks)

## My survey sites - dataframe
my_sites_df <- read.csv("data/basic_intro/survey_sites.csv")
my_sites    <- st_as_sf(x = my_sites_df, coords = c("long", "lat"), crs = 4326) # EPSG for WGS84, what my gps was set in.

my_sites


## for plotting, state boundary
vic <- st_read("data/basic_intro/vic.gpkg")

```

CRS and projections

Coordinate reference systems (CRS)

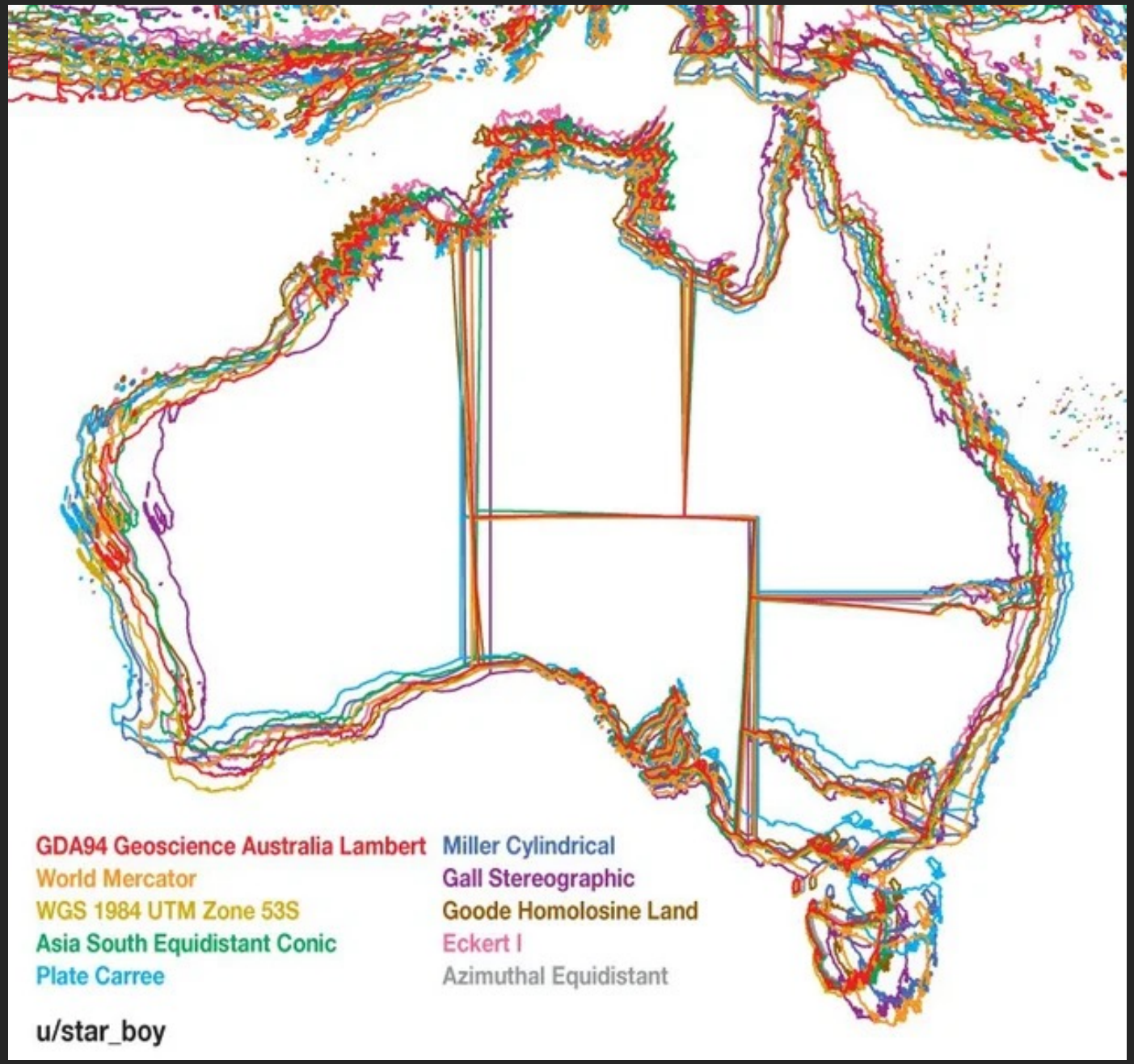
- https://ihatecoordinatesystems.com/

Geographic vs projected

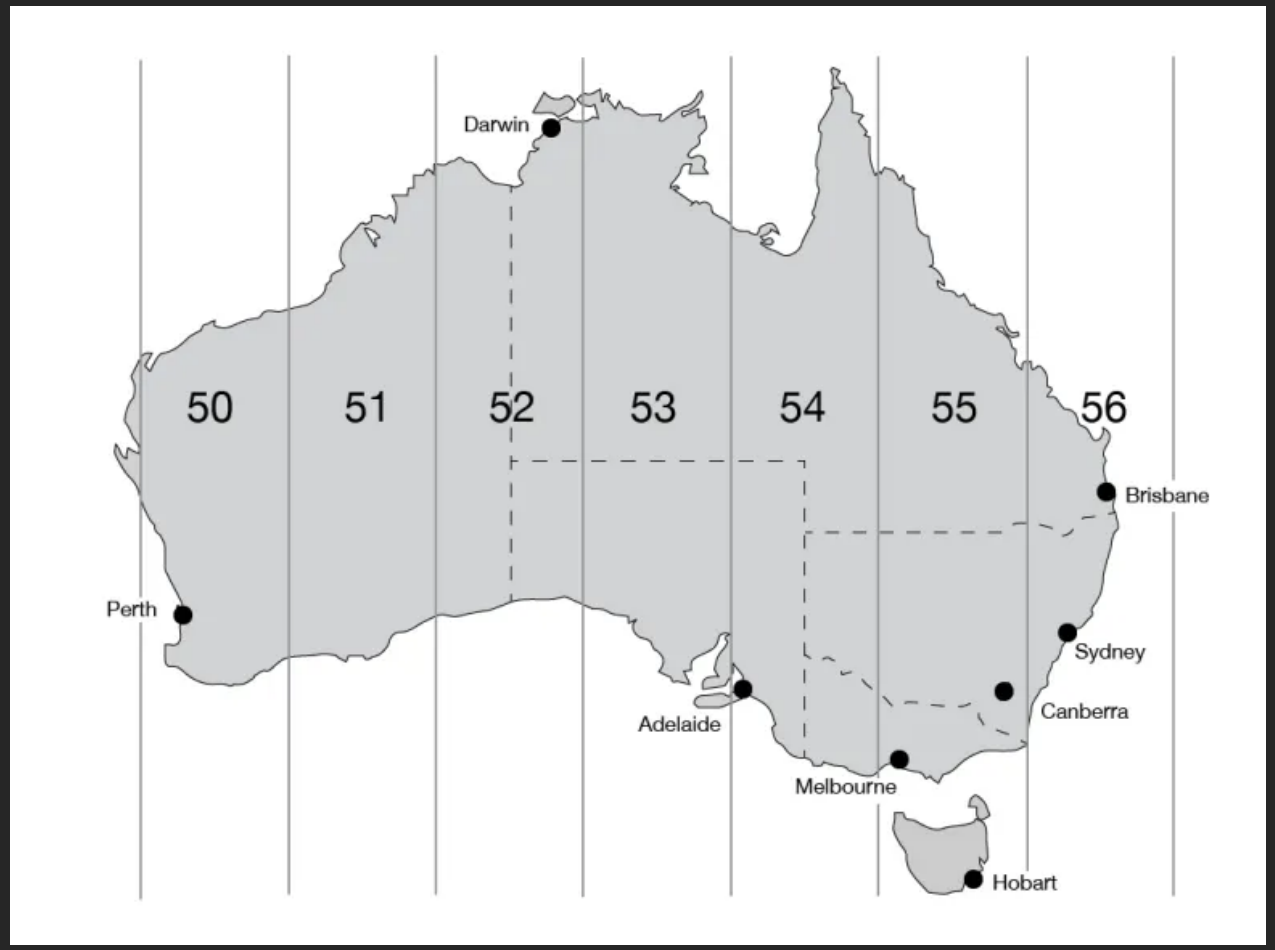
- for sf, use a projected crs for your area
  - e.g. WGS84 (global, non-projected). Melbourne 144.96°, -37.81°
  - E.g. for Australia, GDA2020 (Australia specific) and MGA, UTM (global, projected)

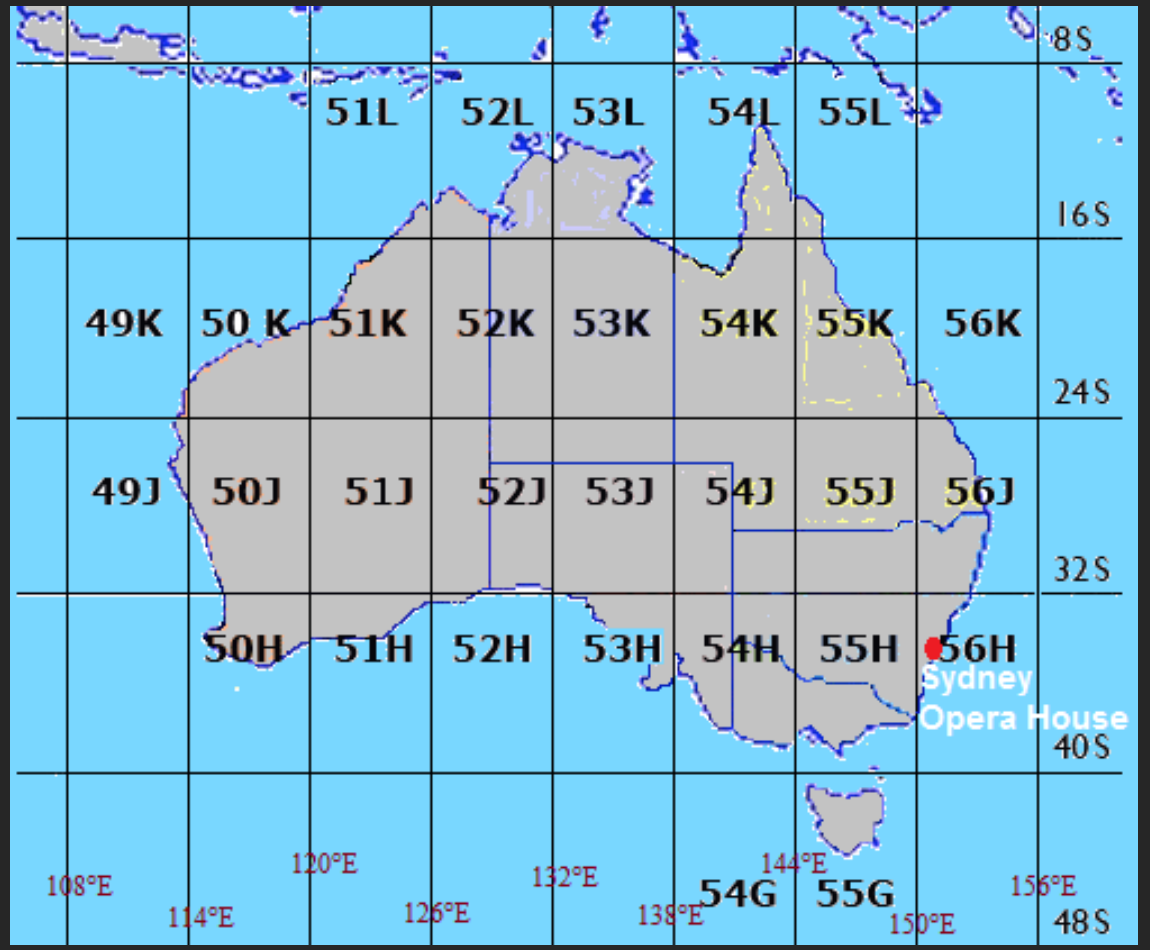
EPSG: A 4 digit unique code for each crs

- E.g. WGS84 = EPSG 4326


GDA94 Geoscience Australia Lambert
World Mercator
WGS 1984 UTM Zone 53S
Asia South Equidistant Conic
Plate Carree
Miller Cylindrical
Gall Stereographic
Goode Homolosine Land
Eckert I
Azimuthal Equidistant
u/star_boy

(UTM)


Darwin
50
51
52
53
54
55
56
Brisbane
Perth
Sydney
Canberra
Adelaide
Melbourne
Hobart


8S
51L
52L
53L
54L
55L
16S
49K
50 K
51K
52K
53K
54K
55K
56K
24S
49J
50J
51J
52J
53J
54J
55J
56J
32S
50H
51H
52H
53H
54H
55H
56H
Sydney
Opera House
40S
108°E
114°E
120°E
126°E
132°E
138°E
144°E
150°E
156°E
54G
55G
48S

In R:

```r
# projections ----------------------------------------------------------
## New functions:
?st_crs
?st_transform
?st_geometry


## our data
parks    # Note: Projected CRS: GDA2020 / MGA zone 55
my_sites # Geodetic CRS:  WGS 84

## Check the crs using st_crs()
st_crs(parks)
st_crs(my_sites)
# note: they are different. need to reproject my_sites from a geographic crs (WGS84) to a projected crs for melbourne (e.g. MGA2020 55)

## two methods:
# 1. using the EPSG, or the crs
my_sites_p <- st_transform(my_sites, crs = 7855)
# 2. extract the crs from another layer that you know is correct
my_sites_p <- st_transform(my_sites, crs = st_crs(parks))

# plot them
plot(st_geometry(parks))                                    # to zoom to the right area
plot(st_geometry(vic),        add = TRUE, col = "lightgrey") # beacuse this is a polygon it plots over the top and covers the parks
plot(st_geometry(parks),      add = TRUE, col = "yellow")    # so add the parks again
plot(st_geometry(my_sites),   add = TRUE, col = "blue")      # dont appear - wrong projection
plot(st_geometry(my_sites_p), add = TRUE, col = "red")


# why use st_geometry ?
# if your data has columns, each will be plotted
# e.g. sites has 2 attributes (check the dataframe to see them) so the points plot twice
plot(my_sites)

```

Use your normal data cleaning and manipulation techniques

In R:

```
93
94  # Normal data cleaning ──────────────────────────────────────────────────────
95  park_of_interest ← parks[parks$NAME == "Albert Park",]
96  park_of_interest ← parks %>% filter(NAME == "Albert Park")
97
98  # plot
99  plot(st_geometry(parks))
100 plot(st_geometry(vic),              add = TRUE, col = "lightgrey")
101 plot(st_geometry(parks),            add = TRUE, col = "yellow")
102 plot(st_geometry(park_of_interest), add = TRUE, col = "green")
103 plot(st_geometry(my_sites_p),       add = TRUE, col = "red")
104
105
```

Common functions – st_join and st_buffer()

St_join

- Transfer attributes between layers where they overlap

St_buffer

- If projected crs, this unit will be in an unit like meters. If its not projected, this would be in degrees
- Around points
- Around polygon

In R:

```r
# Joining attributes ----------------------------------------------------------
## New functions:
?st_join

# for my sites, as the data from the park layer where the sites are
my_sites_dat <- st_join(my_sites_p, parks)
my_sites_dat

# Add the data from my sites to the parks layer
my_parks_dat <- st_join(parks, my_sites_p)
my_parks_dat # see the NAs for the park polygons where I have no sites intersecting


# Buffering -------------------------------------------------------------------
## New functions:
?st_buffer

# around a polygon:
park_oi_buf <- st_buffer(park_of_interest, dist = 2000)

# around a point:
my_sites_buf <- st_buffer(my_sites_dat, dist = 500)

# compare: the point and the buffered points (now polygons)
my_sites_dat
my_sites_buf


plot(st_geometry(parks))
plot(st_geometry(vic),              add = TRUE, col = "lightgrey")
plot(st_geometry(parks),            add = TRUE, col = "yellow")
plot(st_geometry(park_of_interest), add = TRUE, col = "green")
plot(st_geometry(my_sites_p),       add = TRUE, col = "red")
# Add the buffers
plot(st_geometry(park_oi_buf),      add = TRUE)
plot(st_geometry(my_sites_buf),     add = TRUE)
```

Common functions – st_intersect and st_intersection

Objects that cross over (intersect)

st_intersect

- Keep the entire object. Returns a list that you use to subset your original data.

st_intersection

- cuts to only the overlapping area. Returns a point, line or polygon

In R:

```r
# Intersecting ----------------------------------------------------------
## New functions:
?st_intersects
?st_intersection

# parks that are within 2km of our park of interest, Albert Park
# Keep the parks that intersect the 2km buffer
parks_intersect_list <- st_intersects(park_oi_buf, parks)
# returns a list
parks_intersect_list
# get the polygons to keep - the ones that intersect
parks_intersect_list[[1]]

parks_intersect <- parks[parks_intersect_list[[1]],]

# parks_non_intersect <- parks[!parks_intersect_list[[1]],]


plot(st_geometry(parks))
plot(st_geometry(vic),              add = TRUE, col = "lightgrey")
plot(st_geometry(parks),            add = TRUE, col = "yellow")
plot(st_geometry(park_of_interest), add = TRUE, col = "green")
plot(st_geometry(park_oi_buf),      add = TRUE)
plot(st_geometry(parks_intersect),  add = TRUE, col = "darkgreen")
```

```r


# Keep parks that are within 500m of our sites
my_site_parks_intersect <- st_intersection(my_sites_buf, parks)
# returns an sf object
my_site_parks_intersect

plot(st_geometry(parks))
plot(st_geometry(vic),                    add = TRUE, col = "lightgrey")
plot(st_geometry(parks),                  add = TRUE, col = "yellow")
plot(st_geometry(my_sites_p),             add = TRUE, col = "red")
plot(st_geometry(my_sites_buf),           add = TRUE)
plot(st_geometry(my_site_parks_intersect), add = TRUE, col = "darkred")

```

Dplyr and sf

- Data cleaning and manipulation package from tidyr

In R:

```r
# dplyr and other functions ------------------------------------------------------------

# Intersecting parks
parks_intersect
# how many parks are within 2km of our park?
nrow(parks_intersect)

# Area
## Functions:
?st_area
st_area(my_site_parks_intersect)

my_parks_full <- my_site_parks_intersect %>% mutate(area = st_area(.))

# Make it a dataframe
my_parks_df <- my_site_parks_intersect %>%
  mutate(area = as.vector(st_area(.))) %>%
  filter(area > 300000) %>%
  select(site_name, area) %>%
  st_drop_geometry()


# Summarise and group_by
parks %>%
  group_by(NAME) %>%
  summarise(n = n())

```

Plotting

1) Base r
   - Add = true, col =

2) Ggplot
   - Good for sf objects
   - geom_sf()

3) Tmap
   - Good for sf objects and rasters together
   - tm_shape()

In R:

```r
# plot ----------------------------------------------------------------------
## New functions:
?geom_sf
?tm_shape
?tm_polygons
?tm_borders
?tm_dots

## base r
plot(my_parks_full)
plot(st_geometry(my_parks_full))



## ggplot
# good for sf objects
ggplot() +
  geom_sf(data = parks) +
  geom_sf(data = my_parks_full, fill = "darkred") +
  geom_sf(data = my_sites_dat, col = "red")



## tmap
# good for rasters and sf objects
tm_shape(my_parks_full) +
  tm_polygons()

tm_shape(parks)           + tm_polygons() +
  tm_shape(vic)           + tm_borders(col = "black") +   # Vic as an outline
  tm_shape(my_parks_full) + tm_polygons(col = "darkred") + # parks that overlap
  tm_shape(my_sites_dat)  + tm_dots(size = 0.1, col = "red") # sites

```

Notes

- Use R for the analysis (more reproducible) and GIS to view the data
- Michael – file structure a geopackages. layers
- Michael – size of the dataframe. Cut it down to only the essential variables

In R:

```
259 # Final - other useful functions ------------------------------------------
260 
261 # Validity
262 ?st_is_valid
263 ?st_make_valid
264 
265 # empty geometry
266 ?st_is_empty
267 
```

Now for a real life ecological example