# Spatial R Exercise 1

Ben Davies

2022-05-08

# Contents

# Chapter 1

# Adding the `sf` package

First, we'll include the `sf` package that we'll use for working with vector data.

```
require(sf)
```

```
## Loading required package: sf
```

```
## Linking to GEOS 3.9.1, GDAL 3.3.2, PROJ 7.2.1; sf_use_s2() is TRUE
```

Remember that this package is built as an update of the `sp` package. This uses the *Simple Features* object format, where spatial data are stored as dataframes with a special "geometry" column. This makes them more flexible for use by other packages. You can read more about Simple Features here, and about the history of spatial data handling in R here.

# Chapter 2

# Working with spatial data

## 2.1 Reading in a shapefile

Now that we have access to the `sf` package, we'll read in a shapefile (south_africa_border.shp) with the borders of the countries in southern Africa: South Africa, Lesotho, and Eswatini.

```
#Import the south_africa_border.shp shapefile.
southernAfrica<-st_read("south_africa_border.shp")
```

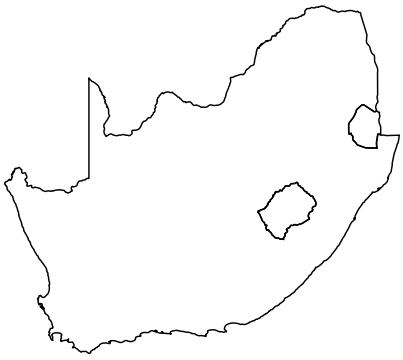
```
## Reading layer `south_africa_border' from data source
##   `C:\Users\bdav_\Dropbox\Teaching\Spatial R Short Course\Bookdown\Exercise1\south_africa_bord
##   using driver `ESRI Shapefile'
## Simple feature collection with 3 features and 94 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 16.46998 ymin: -34.82195 xmax: 32.89308 ymax: -22.12645
## Geodetic CRS:  WGS 84
```

This dataset comes from Natural Earth, and it includes 95 variables about these 3 countries, including population, GDP, country names in other languages, etc. If you wanted to take a look at it in more detail, you could use `str(southernAfrica)`, but you can also take my word for it.

## 2.2 Plotting spatial data

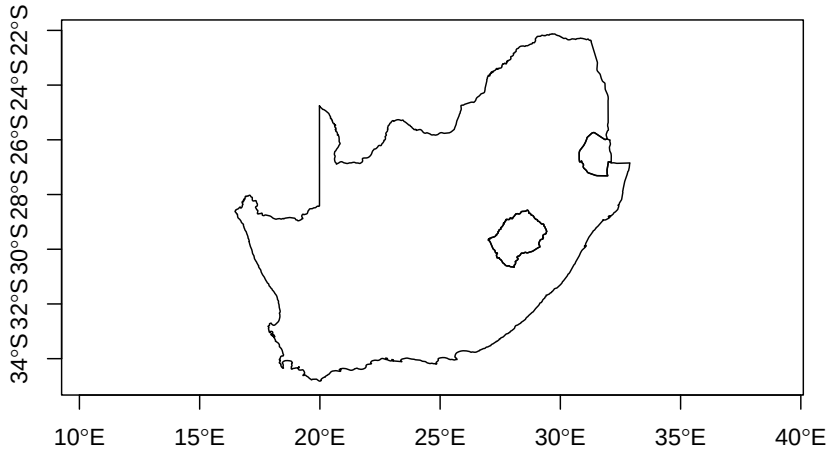
Next we'll use `plot` and `st_geometry` to plot the borders.

```
#Plot geometry
plot(st_geometry(southernAfrica))
```



While we all probably have a sense of where southern Africa is, it's usually a good idea to include some information about the geography. We'll learn more about making maps later, but for now we can just include the axes to give us latitude and longitude.

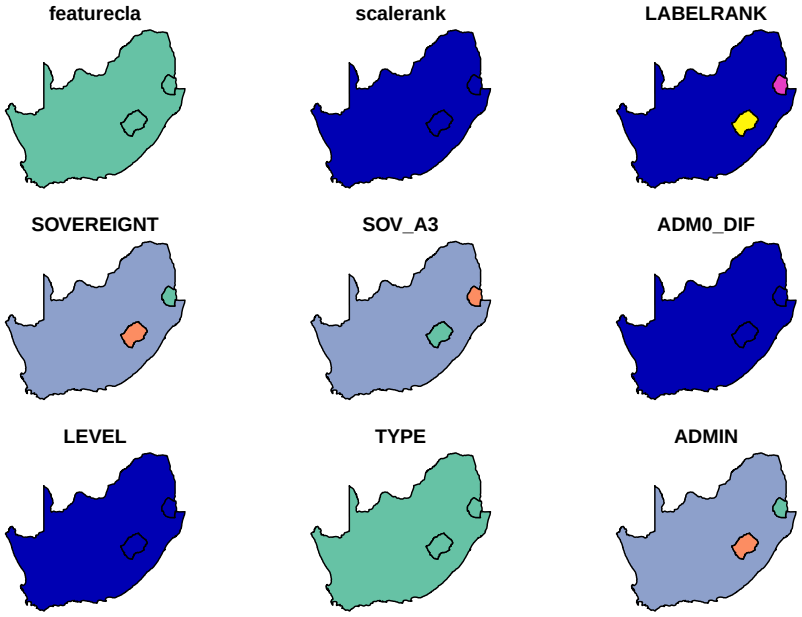
```
#Plot geometry with axes
plot(st_geometry(southernAfrica),axes=T)
```




If we want to see the information in the dataset, we could just use the `plot` function by itself.

```
plot(southernAfrica)
```

```
## Warning: plotting the first 9 out of 94 attributes; use max.plot = 94 to plot
## all
```




You may notice that this gives us the first nine variables (or columns) in the dataset. Since a vector dataset might contain hundreds of variables, the `sf` package will stop after 9 plots by default so that R doesn't freak out. You may also notice that for many of these, there's no differences between these countries. It's probably more useful to target a specific variable of interest. For example, if we want to access the population estimate for each country and plot that, we first need to figure out which variable corresponds to the population. You can get a list of variable names using `colnames` or just by typing `southernAfrica$` into the command line.

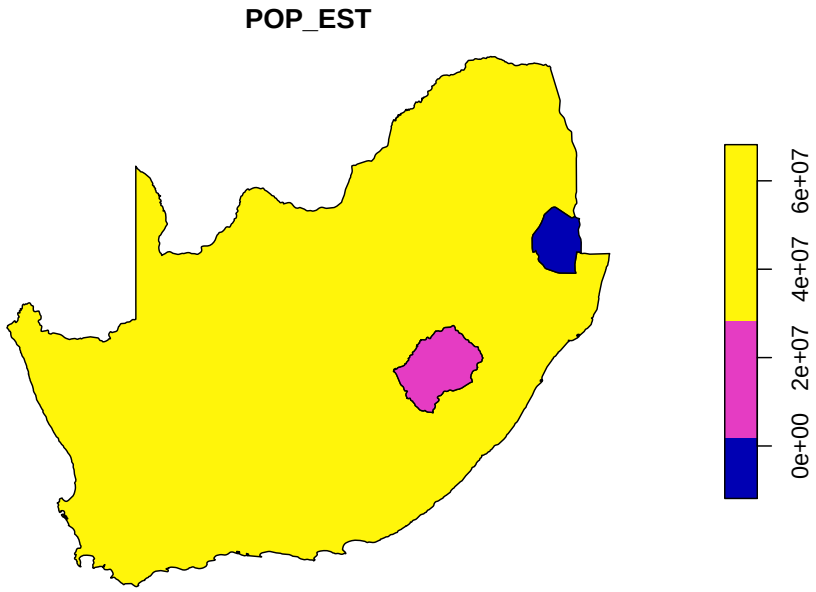
```
colnames(southernAfrica)
```

```
##  [1] "featurecla" "scalerank"  "LABELRANK"  "SOVEREIGNT" "SOV_A3"
##  [6] "ADM0_DIF"   "LEVEL"      "TYPE"       "ADMIN"      "ADM0_A3"
## [11] "GEOU_DIF"   "GEOUNIT"    "GU_A3"      "SU_DIF"     "SUBUNIT"
## [16] "SU_A3"      "BRK_DIFF"   "NAME"       "NAME_LONG"  "BRK_A3"
## [21] "BRK_NAME"   "BRK_GROUP"  "ABBREV"     "POSTAL"     "FORMAL_EN"
## [26] "FORMAL_FR"  "NAME_CIAWF" "NOTE_ADM0"  "NOTE_BRK"   "NAME_SORT"
## [31] "NAME_ALT"   "MAPCOLOR7"  "MAPCOLOR8"  "MAPCOLOR9"  "MAPCOLOR13"
## [36] "POP_EST"    "POP_RANK"   "GDP_MD_EST" "POP_YEAR"   "LASTCENSUS"
## [41] "GDP_YEAR"   "ECONOMY"    "INCOME_GRP" "WIKIPEDIA"  "FIPS_10_"
## [46] "ISO_A2"     "ISO_A3"     "ISO_A3_EH"  "ISO_N3"     "UN_A3"
## [51] "WB_A2"      "WB_A3"      "WOE_ID"     "WOE_ID_EH"  "WOE_NOTE"
## [56] "ADM0_A3_IS" "ADM0_A3_US" "ADM0_A3_UN" "ADM0_A3_WB" "CONTINENT"
## [61] "REGION_UN"  "SUBREGION"  "REGION_WB"  "NAME_LEN"   "LONG_LEN"
```

```
## [66] "ABBREV_LEN" "TINY"       "HOMEPART"    "MIN_ZOOM"    "MIN_LABEL"
## [71] "MAX_LABEL"  "NE_ID"      "WIKIDATAID" "NAME_AR"     "NAME_BN"
## [76] "NAME_DE"    "NAME_EN"    "NAME_ES"     "NAME_FR"     "NAME_EL"
## [81] "NAME_HI"    "NAME_HU"    "NAME_ID"     "NAME_IT"     "NAME_JA"
## [86] "NAME_KO"    "NAME_NL"    "NAME_PL"     "NAME_PT"     "NAME_RU"
## [91] "NAME_SV"    "NAME_TR"    "NAME_VI"     "NAME_ZH"     "geometry"
```

There's a lot of variables here, but once we know the name of the variable we are interested in, we can plot this using brackets to include the variable name:

```
plot(southernAfrica['POP_EST'])
```




This shows us that, unsurprisingly, there are population differences between South Africa, Lesotho, and Eswatini. But more importantly, it shows that you can import and explore this data using a few lines of code.

## 2.3 Try it yourself!

Add some code to see if you can do the following with the southern Africa data:

- Figure out which variable/column gives the gross domestic product (GDP) estimate for each country
- Plot the GDP data with latitude/longitude axes

- Can you think of a way to plot the population estimate and GDP side by side (with axes)?

# Chapter 3

# Subsetting data

## 3.1 Adding more data

In this step, we'll import some more slightly more interesting data: biomes in southern Africa.

```
#Import SABiomes.csv data
biomes<-st_read("SABiomes.shp")
```

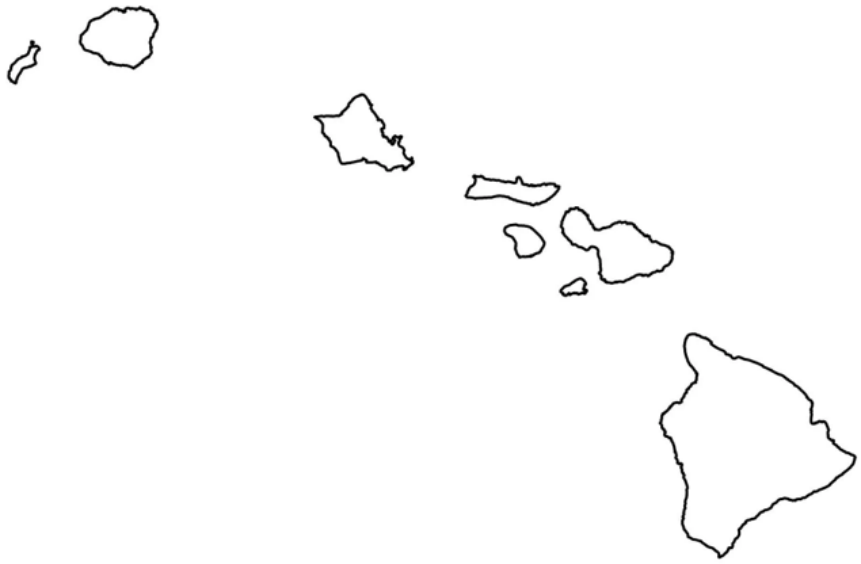
```
## Reading layer `SABiomes' from data source
##   `C:\Users\bdav_\Dropbox\Teaching\Spatial R Short Course\Bookdown\Exercise1\SABiomes.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 287 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 16.45696 ymin: -34.8334 xmax: 32.89179 ymax: -22.12917
## Geodetic CRS:  WGS 84
```

This dataset comes from the South Africa National Biodiversity Institute and is based on a 2006 classification scheme by Mucina and Rutherford.

You might notice that when you import this dataset, it reports the following:

`Geometry type: MULTIPOLYGON`

The "multi" types of geometry isn't something we have seen yet, but you may be familiar with these if you have used other GIS platforms. The main difference between a `POLYGON` and `MULTIPOLYGON` geometry is that in a `POLYGON` geometry, each individual polygon is a distinct feature, while in a `MULTIPOLYGON` geometry, multiple polygons might be included in a single feature. You could think about it in terms of the main islands of Hawaii:



APOLYGON dataset might have separate features for Oahu, Maui, Kauai, and so on, while a MULTIPOLYGON dataset might have 8 polygons that are all a single feature called "Hawaii". For the biomes dataset, some areas may include disconnected components, but are here considered to be a single feature or "multipolygon".

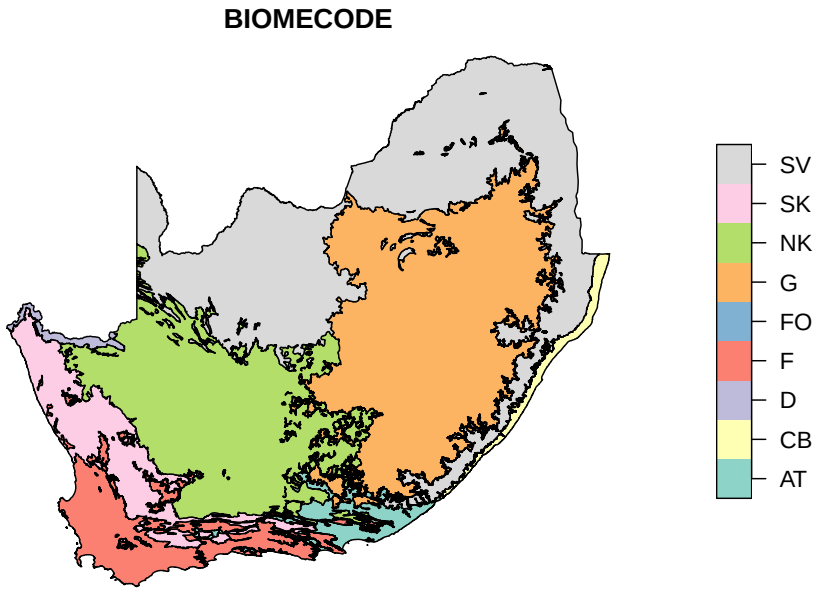
Let's use the unique function to take a look at how many different biomes are in this dataset:

```
#How many features are in this dataset?
unique(biomes$BIOMENAME)
```

```
## [1] "Forests"                    "Grassland"
## [3] "Savanna"                    "Nama-Karoo"
## [5] "Desert"                     "Fynbos"
## [7] "Indian Ocean Coastal Belt" "Succulent Karoo"
## [9] "Albany Thicket"
```

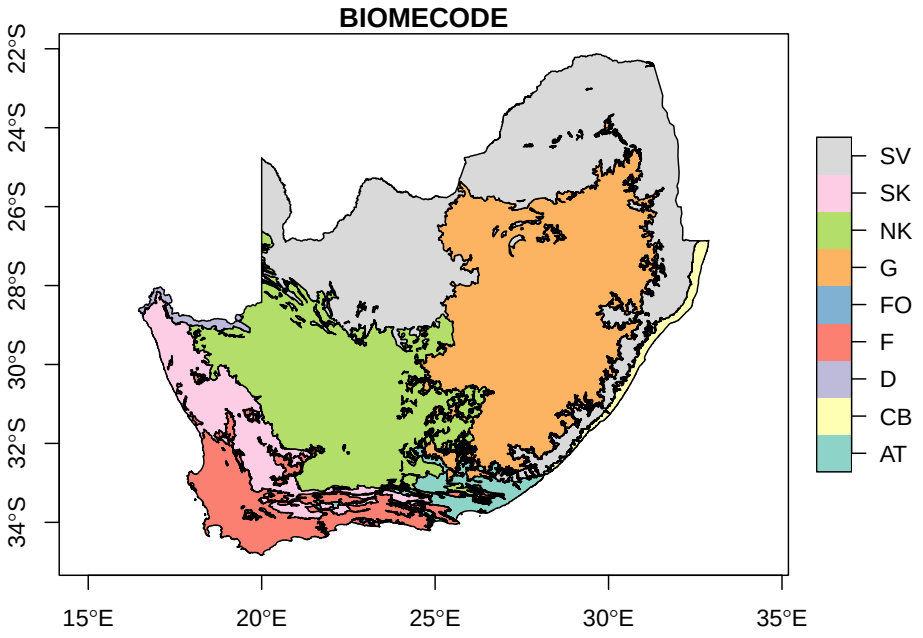
This shows us the names of all 9 of the primary biomes in South Africa. Now we can look at their distribution by plotting them. We probably don't need to plot every variable, so let's just use the BIOMECODE variable to plot by.

```
#Plot by biome code
plot(biomes['BIOMECODE'])
```




Lovely! We should also add axes.

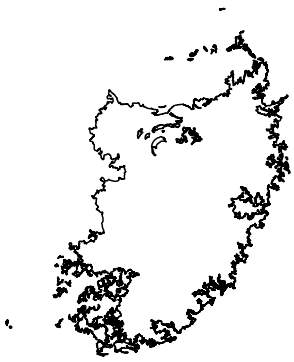
```
#Plot by biome code with axes
plot(biomes['BIOMECODE'],axes=T)
```



## 3.2 Subsetting using attribute data

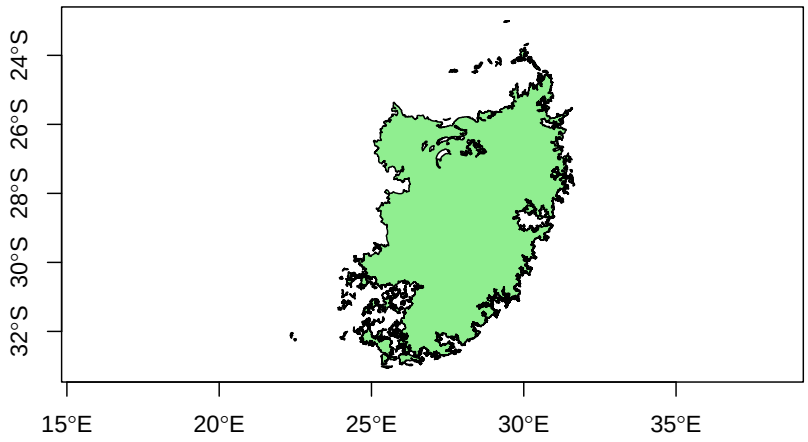
Now let's say we're only intrested in the grassland areas. We can use `subset` to reduce the dataset just to grassland areas.

```
#Plot just grasslands
grassland<-subset(biomes,BIOMENAME=="Grassland")
plot(st_geometry(grassland))
```



Again, let's include the axes, and let's also make the color light green.

```
#Print the grasslands green with axes
plot(st_geometry(grassland),axes=T,col="light green")
```




Looking good so far!

## 3.3 Try it yourself!

Modify the blank code chunk below to see if you can do the following with the biomes data:

- Subset the data so that it is just the savanna biome
- Plot this and choose a complementary color. If you're not sure what the options are for colors, try this document.
- Change the type and thickness of the lines used to draw the features with the `lty` and `lwd` arguments. These take numeric values and default to 1. See what happens when you change these.
- See if you can plot the geometry of the southern Africa, and then add the savanna.

# Chapter 4

# Creating `sf` objects from table data

## 4.1 Adding CSV data

Now let's say we have some point data and we want to add it to the map. However, in this case we've got it stored as a comma-separated values (CSV) file. Let's use `read.csv` to read the data in, and `class` to look at the object class.

```
#Import the SARadiocarbon.csv data
c14table<-read.csv("SARadiocarbon.csv")
class(c14table)
```

```
## [1] "data.frame"
```

These data are radiocarbon dates from the South African Radiocarbon Database. The `read.csv` function reads this in as a dataframe only, so it isn't being treated as an sf object just yet. Let's take a look at the dataframe using the `head` function, which gives a preview of the first 6 rows:

```
#Take a look at the first 6 rows
head(c14table)
```

```
##         SITE SITE_CODE LATITUDE LONGITUDE      COUNTRY PROVINCE_DISTRICT RFZ
## 1 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
## 2 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
## 3 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
```

```
## 4 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
## 5 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
## 6 Thula Mela       TUM  -22.431    31.194 South Africa           Limpopo   S
##           BIOME CONTEXT        TECHNIQUE MATERIAL      SPECIES   LAB_ID DATE
## 1 Savanna Biome         conventional 14C charcoal              Pta-6079  480
## 2 Savanna Biome         conventional 14C charcoal              Pta-7107  530
## 3 Savanna Biome         conventional 14C charcoal              Pta-7103  390
## 4 Savanna Biome         conventional 14C charcoal              Pta-7105  560
## 5 Savanna Biome         conventional 14C     bone Homo sapiens Pta-7276  370
## 6 Savanna Biome         conventional 14C     bone Homo sapiens Pta-7243  520
##   UNCERTAINTY ARCH_PERIOD ARCH_SUBPERIOD SAMPLE_CONTEXT  SITE_TYPE DELTA_13
## 1          30    Iron Age                                settlement       NA
## 2          30    Iron Age                                settlement       NA
## 3          35    Iron Age                                settlement       NA
## 4          35    Iron Age                                settlement       NA
## 5          40    Iron Age                                settlement       NA
## 6          40    Iron Age                                settlement       NA
##        REFCODE   NOTES
## 1 vogel2000rds   Khami
## 2 vogel2000rds Mutamba
## 3 vogel2000rds   Khami
## 4 vogel2000rds Mutamba
## 5 vogel2000rds   Khami
## 6 vogel2000rds   Khami
```

Among the columns in this dataset is one called `BIOME`, which tells us from which biome the date was recovered. There are more than 2000 dates in the set, but there probably aren't that many biomes in the dataset. Again, we can use `unique` to get a list of the ones included here.

```
#Get unique biome names
unique(c14table$BIOME)
```

```
##  [1] "Savanna Biome"             "Grassland Biome"
##  [3] "Succulent Karoo Biome"     "Azonal Vegetation"
##  [5] "Desert Biome"              "Nama-Karoo Biome"
##  [7] "Fynbos Biome"              "Thicket Biome"
##  [9] "Forests"                   "Indian Ocean Coastal Belt"
## [11] ""
```

While there are some minor differences, these match up pretty well with the names from our biomes shapefile. If we wanted just the dates from the grasslands, we could subset the data using this column.

```
#Get the grassland dates
c14grass<-subset(c14table,BIOME=="Grassland Biome")
unique(c14grass$BIOME)
```

```
## [1] "Grassland Biome"
```

Terrific. Now we want to make this usable with `sf`. Lucky for us this is already spatial data: coordinates are stored in the columns `LONGITUDE` and `LATITUDE`. We can use these columns as arguments with the `st_as_sf` to coerce the dataframe to a point dataset.

```
#Convert to point object
c14points<-st_as_sf(c14grass,coords=c("LONGITUDE","LATITUDE"),remove=FALSE)
```

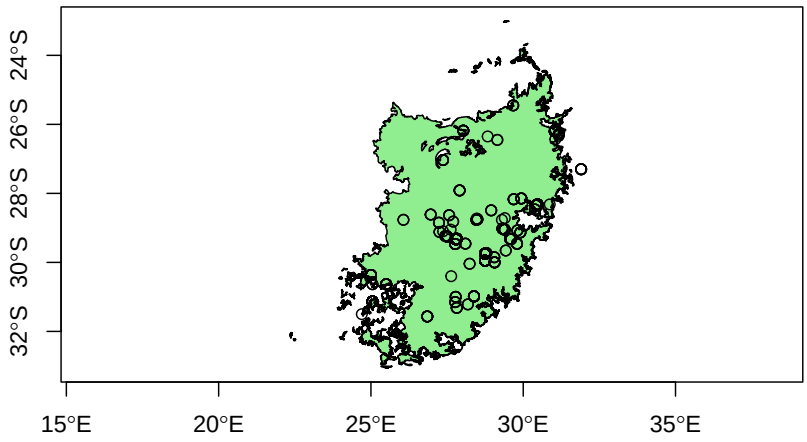
Notice that we added an argument to the `st_as_sf` function that says `remove=FALSE`. This means that when we use `LONGITUDE` and `LATITUDE` to generate our geometry, we don't remove these columns. These will come in handy later on. For now, let's check on the class of this new object.

```
class(c14points)
```

```
## [1] "sf"         "data.frame"
```

OK, now it's in the sf class, so we can add it to our map.

```
#Add to plot
plot(st_geometry(grassland),axes=T,col="light green")
plot(st_geometry(c14points),add=T)
```




## 4.2 Subsetting using location data

Not bad! Notice that one point just east of the edge of the grassland boundary? It's not clear what's happened there: it could be from a site that is in a transition zone, or maybe was mislabeled by the data authors. Whatever it is, let's say we don't want to include that point. We can use subset again, but this time we only want to include points that are inside our grassland object. We'll use `st_bbox` to get the extent of the grassland object:

```
st_bbox(grassland)
```

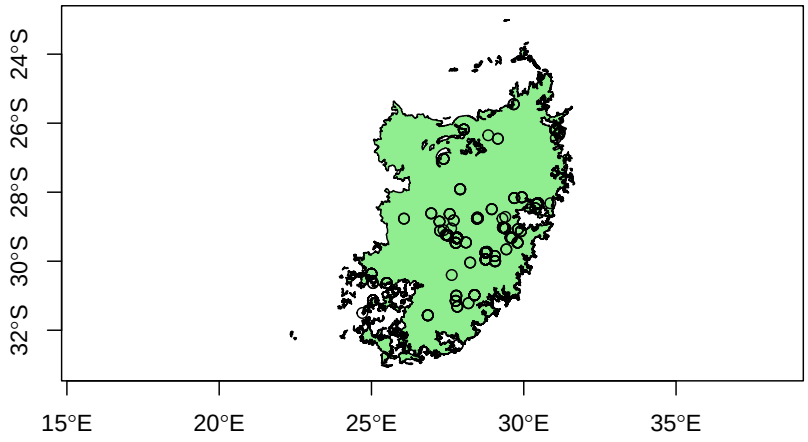
```
##      xmin      ymin      xmax      ymax
##  22.33500 -33.06480  31.66951 -22.99997
```

These values are the maximum and minimum x and y coordinates for the grassland object. We can see that the `xmax`, or the longitude that is the furthest to the east, is 31.66951. We can use `subset` and the `LONGITUDE` column we kept to pare it down.

```
c14points2<-subset(c14points, LONGITUDE < 31.66951)
```

And then we plot again..

```
#Add to plot
plot(st_geometry(grassland),axes=T,col="light green")
plot(st_geometry(c14points2),add=T)
```




Fixed! This is an example of *spatial subsetting*. In this case, since there was only one point we wanted to prune and we had the `LONGITUDE` data handy, it was pretty easy to do it this way. But if the , we'd probably want to find a way to crop the points using the biome shape itself. This is something we'll cover in more detail in another session.

# Chapter 5

# Creating sf objects from scratch

## 5.1 Making spatial data

Lastly, let's add some data we make ourselves. Let's say we have a section of grassland we want to use as a study area. To do this, we're going to make a square polygon using longitude/latitude coordinates. First, we use `list` and `rbind` to create a list object with a matrix of coordinate pairs:

```
#Make polygon list
polygonList = list(rbind(c(26.5, -28), c(29.5, -28), c(29.5, -30), c(26.5, -30), c(26.5, -28)))
```

Next, we trainslate this list into a polygon object with `st_polygon`, and then a feature collection with `st_sfc`:

```
polygonObject<-st_polygon(polygonList)
polygonCollection<-st_sfc(polygonObject)
```

Now we generate some "data" to include. Since it's only one feature we're making, we can just give it a feature number of 1. We turn this into a dataframe, and then combine them with the feature collection to make a new simple features dataset with `st_sf`:

```
featureNum<-c(1)
data<-data.frame(featureNum)
polygonData<-st_sf(data,polygonCollection)
```

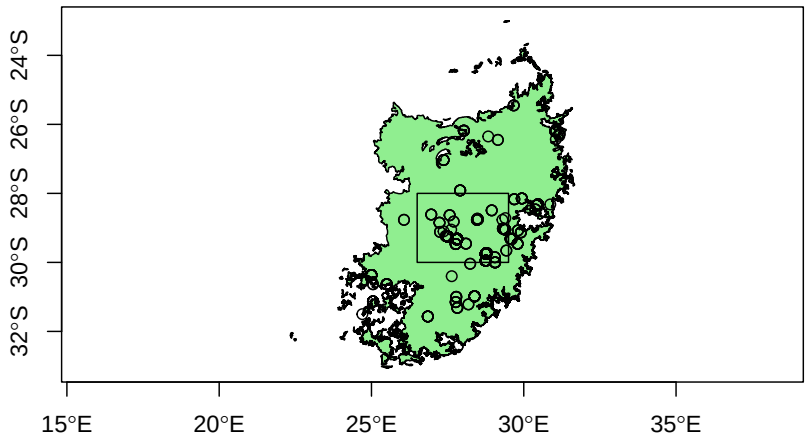
Finally, we need to tell it what coordinate system we are using. The EPSG coordinate system code for the WGS84 Geographic Coordinate System (the standard used in GPS) is 4326. We can assign this easily using `st_crs`, and use the same function to check it:

```
st_crs(polygonData) <- 4326
st_crs(polygonData)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   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["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Horizontal component of 3D system."],
##         AREA["World."],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

Lastly, we want to plot our newly created study area polygon with our other data:

```
#Add to plot
plot(st_geometry(grassland),axes=T,col="light green")
plot(st_geometry(c14points2),add=T)
plot(st_geometry(polygonData),add=T)
```




## 5.2 Writing to file

And if we want to export our new data as a shapefile, we can use `st_write`:

```
#Export it to shapefile (remove the # to run this code)
#st_write(polygonData,"sampleArea.shp",driver="ESRI shapefile")
```

# Chapter 6

# Bring it all together!

Try to write code to do the following on your own:

- Load the biomes shapefile and radiocarbon data
- Subset both of these to the Fynbos Biome
- Make a histogram using the dates in the fynbos subset of the radiocarbon data using the `hist` function
- Create another subset of the fynbos radiocarbon data that includes Pleistocene dates only (dates older than 11700 BP)
- Plot the following:
  - The fynbos biome in the color of your choosing (not red or black) with axes
  - The fynbos radiocarbon dates as small, black circles (you can change the size of the points using the `cex` argument)
  - Pleistocene fynbos radiocarbon dates as larger, red filled circles (you can change the symbols using the `pch` argument)