Learning *R*

Carl James Schwarz

StatMathComp Consulting by Schwarz
cschwarz.stat.sfu.ca @ gmail.com

INTRODUCTION to spatial data

Table of Contents I

CAUTION
THIS IS A VERY COMPLEX FIELD
THIS IS A VERY BRIEF INTRODUCTION

Suggested References:

- `https://geocompr.robinlovelace.net/spatial-class.html#intro-sf`
- Vignettes in the *sf* package.

Spatial Data - Introduction

What is spatial data?

- Data is associated with location.
- Locations described by co-ordinates in a co-ordinate reference system.
- Co-ordinate reference systems map co-ordinates (e.g. long/lat) to earth
- Projections map co-ordinate reference system to flat paper.

Types of spatial data

- vector - points, lines, polygons; with attributes
- raster - pixels - typically from remote sensing and not formed into features.

This introduction will cover VECTOR data only.

Spatial Data - Introduction

What packages?

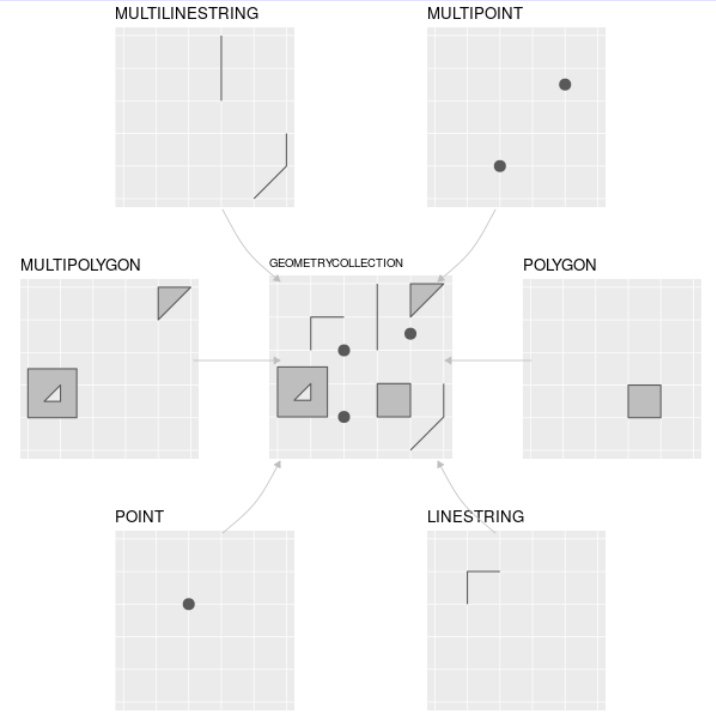
- *sp* - well established but being supplanted by
- *sf* - more consistent; easier to use; supersedes many packages (e.g. *rgeos*, *rdgal*)
- Integration with *ggplot2*
- Easy to use standard data.frame operations etc.

Recommend that you learn the *sf* and if necessary convert to *sp* formats if needed.

Spatial Data - *sf* package

sf = *Simple Features* = open standard.

- Open standard, i.e. not proprietary
- Hierarchical data model
- 17 geometry types, but typically only 7 used



Spatial Data - *sf* package

What is a feature

- A baseline geometry
- A collection of features.

What dimensions are supported?

- XY - traditional long/lat or UTM (notice that longitude is first)
- XYZ - includes height
- XYZM - includes times

We are only looking at XY dimensions in this introduction.

How *sf* are organized I

- All functions start with *st_* (space_time)
- Standard S3 classes, lists, matrices, vectors, data frames
  - Attributes stored in data.frame (or tibble)
  - One column is a *list column* contains the feature feometries
- Integration with *ggplot2* package with *geom_sf()*

```
1 library(spData)
2 head(world)
```

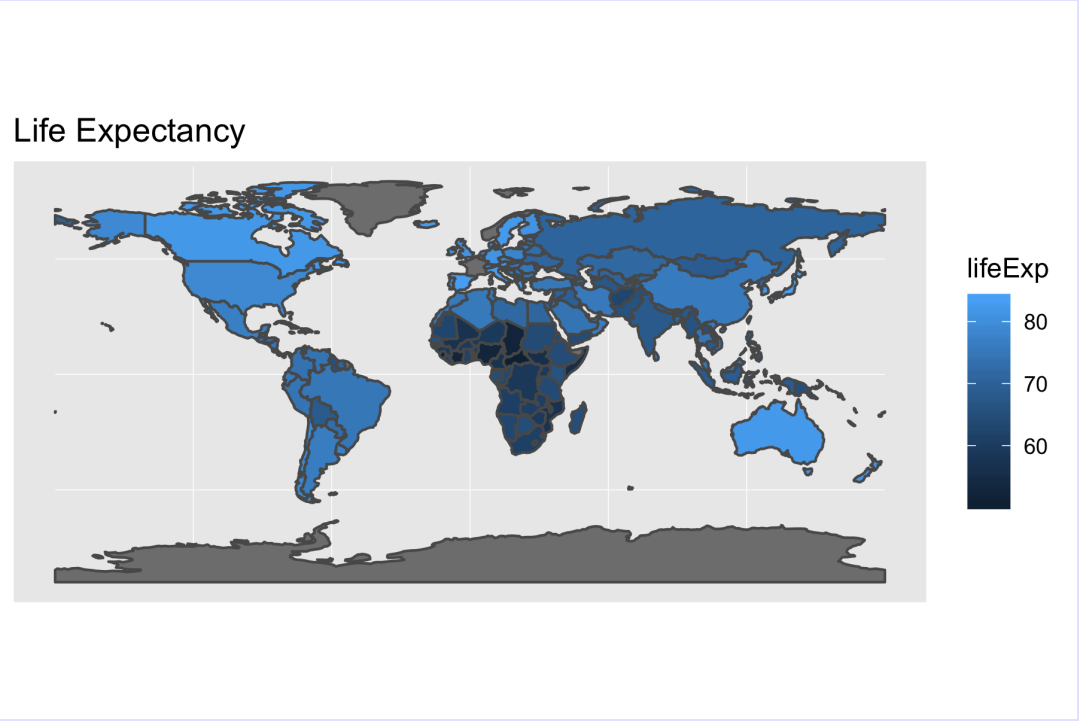
```
> head(world)
Simple feature collection with 6 features and 10 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -180 ymin: -18.28799 xmax: 180 ymax: 8
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

How *sf* are organized II

```
  iso_a2      name_long     continent region_un        subre
1     FJ           Fiji       Oceania   Oceania        Melan
2     TZ       Tanzania        Africa    Africa   Eastern Af
3     EH Western Sahara        Africa    Africa  Northern Af
4     CA         Canada North America  Americas Northern Ame
5     US  United States North America  Americas Northern Ame
6     KZ     Kazakhstan          Asia      Asia     Central
   lifeExp gdpPercap                          geom
1 69.96000  8222.254 MULTIPOLYGON (((180 -16.067...
2 64.16300  2402.099 MULTIPOLYGON (((33.90371 -0...
3       NA        NA MULTIPOLYGON (((-8.66559 27...
4 81.95305 43079.143 MULTIPOLYGON (((-122.84 49,...
5 78.84146 51921.985 MULTIPOLYGON (((-122.84 49,...
6 71.62000 23587.338 MULTIPOLYGON (((87.35997 49...
```

Spatial Data - Plotting

```
1  plot1 <- ggplot()+
2     geom_sf(data=world, aes(fill=lifeExp))+
3     ggtitle("Life Expectancy")
4  plot1
```

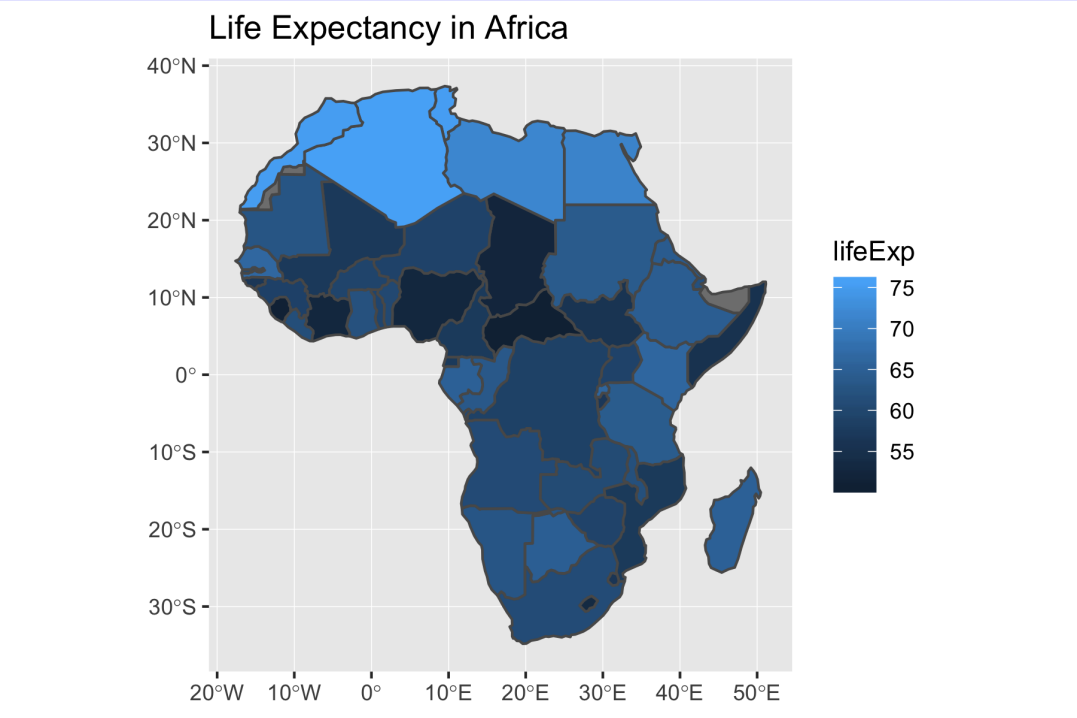


Projection used based on first feature.

Spatial Data - Plotting

Selecting features to plot is straight-forward.

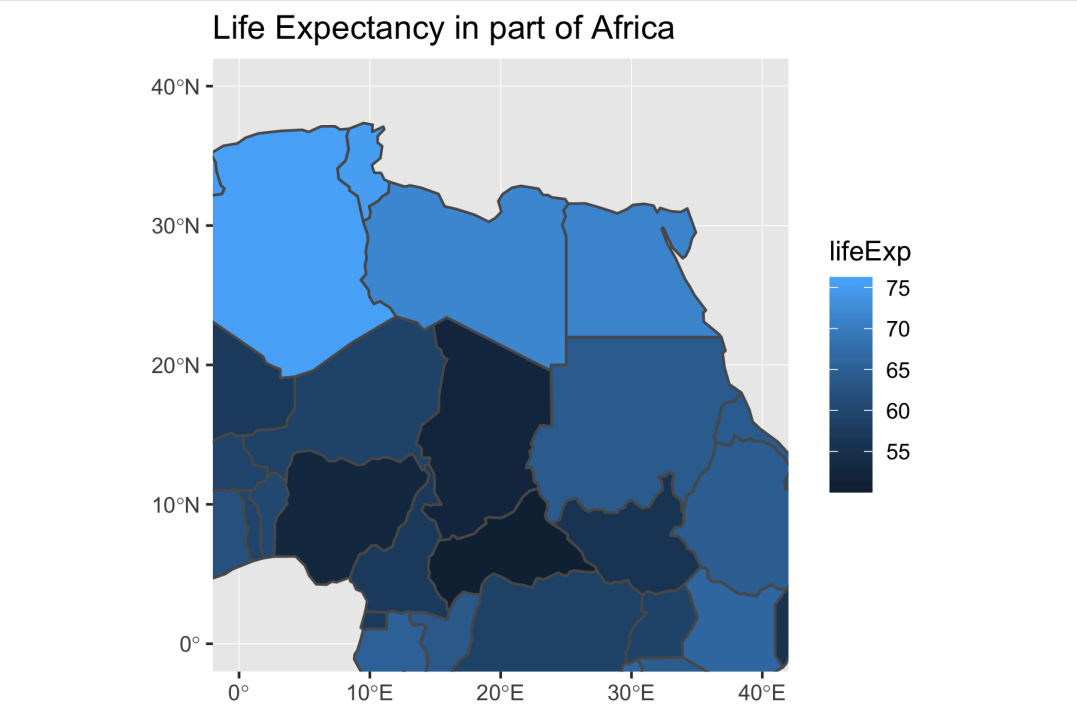
```
1  africa <- world[ world$continent=="Africa",]
2  plot2 <- ggplot()+
3     geom_sf(data=africa, aes(fill=lifeExp))+
4     ggtitle("Life Expectancy in Africa")
5  plot2
```



Projection used based on first feature.

Clipping the map

```
1 plot3 <- ggplot()+
2     geom_sf(data=africa, aes(fill=lifeExp))+
3     ggtitle("Life Expectancy in part of Africa")+
4     xlim(0,40)+ylim(0,40)
5 plot3
```



Projection used based on first feature.

Spatial Data - Creating simple features I

It is possible (but rare) to create your own simple features.

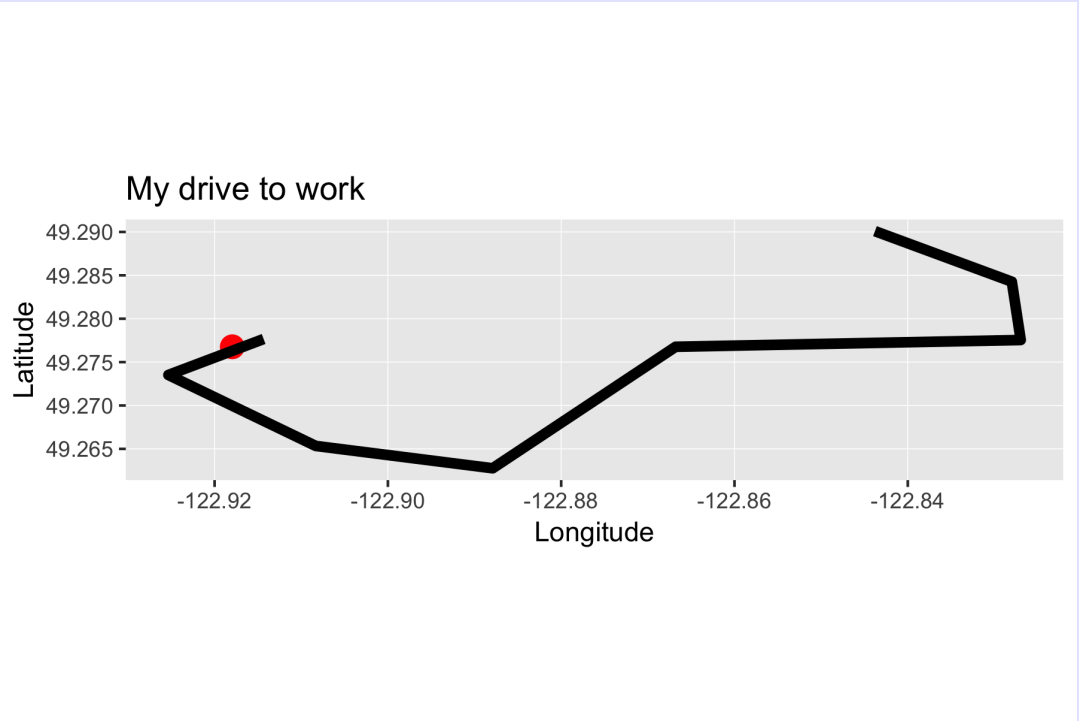
- *st_point()*
- *st_multipoint()*
- *st_linestring()*
- *st_polygon()* - must be closed
- etc.

```
SFU.sf <- sf::st_point(  c(-122.917957, 49.276765 ))
str(SFU.sf)

my.drive.csv <- textConnection("
long, lat
-122.84378900000002, 49.29009199999999
-122.82799615332033, 49.28426960031931
-122.82696618505861, 49.27755059244836
-122.86679162451173, 49.27676664856581
-122.88790597387697, 49.26276555269492
```

Spatial Data - Creating simple features II

```
11  -122.90833367773439, 49.26534205263451
12  -122.92532815405275, 49.27351874831076 4
13  -122.91434182592775, 49.27766258341439")
14  my.drive <- read.csv(my.drive.csv, header=TRUE, as.is=TRUE,
15
16  my.drive.sf <- sf::st_linestring(as.matrix(my.drive[, c("lo
17  str(my.drive.sf)
18
19  plot1 <- ggplot() +
20           ggtitle("My drive to work")+
21           geom_sf(data=SFU.sf, color="red", size=4)+
22           geom_sf(data=my.drive.sf, color="black", size=2, i
23           ylab("Latitude")+xlab("Longitude")
24  plot1
```

Spatial Data - Creating simple features III



Spatial Data - Creating simple features I

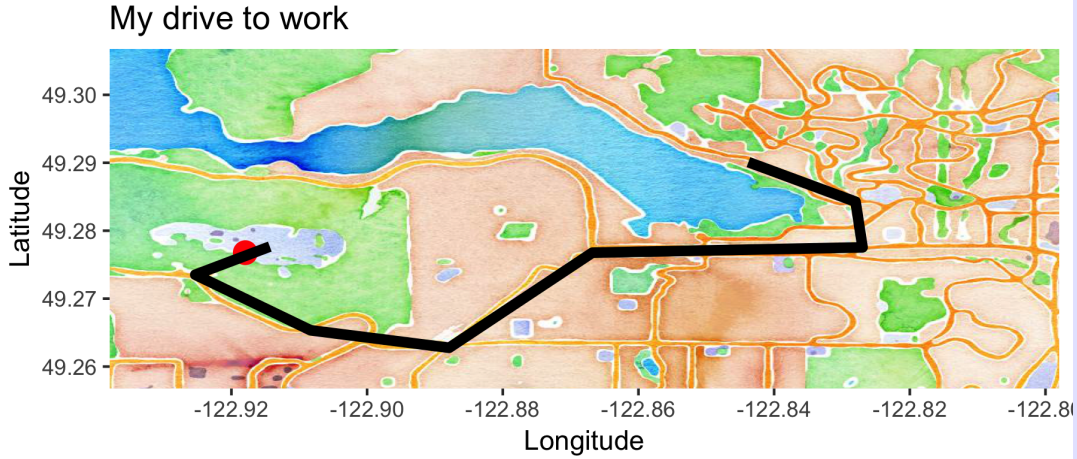
It is possible (but rare) to create your own simple features.
Integrate with *ggmap*

```
1  library(ggmap)
2  sfu.coord <- c(-122.917957, 49.276765 )
3  # get the map from stamen. You can fiddle with the zoom to
4  my.map.dl <- ggmap::get_map(c(left=sfu.coord[1]-.02, bottom=
5                  maptype="watercolor",  source="stamen")
6  my.map <- ggmap(my.map.dl)
7
8  # careful, ggmap uses lon/lat but sf uses long/lat
9  # you need ignore the aes from the {\it ggmap}
10 plot1 <- my.map +
11           ggtitle("My drive to work")+
12           geom_sf(data=SFU.sf, color="red", size=4, inherit.a
13           geom_sf(data=my.drive.sf, color="black", size=2, in
14           ylab("Latitude")+xlab("Longitude")
15 plot1
```

Spatial Data - Creating simple features II

NOTE: Use of *aes.inherits* argument in the *ggplot* code.

Spatial Data - Creating simple features III



Spatial Data - Exercise 1

Refer to accident data base.

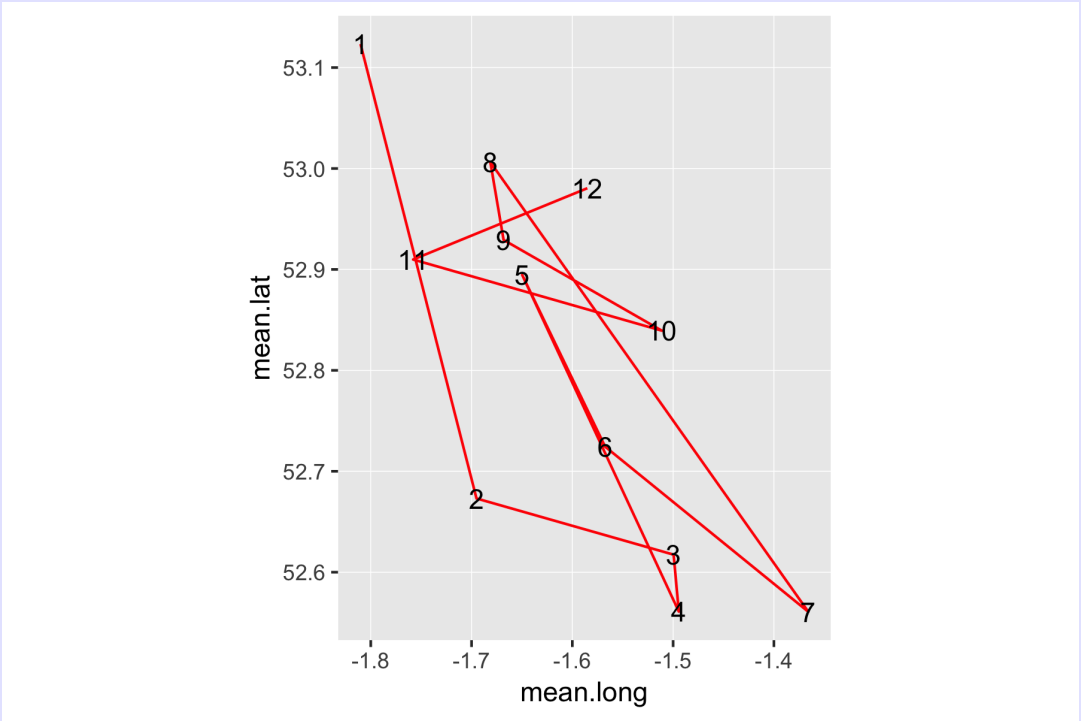
- Select only fatal accidents.
- Find the mean location of fatal accidents by month
- Plot the mean over the year
- Add in background map.

Spatial Data - Exercise 1 I

Note that it is SILLY to compute the mean longitude/latitude (why?), but for simplicity we will do it here.

```
1  library(dplyr)
2  mean.fatal.location <-
3    accidents %>%
4      filter( Fatality==1) %>%
5        group_by(month) %>%
6          summarize( mean.long=mean(Longitude), mean.lat=mean
7  mean.fatal.location
8
9  mean.fatal.location.path.sf <- sf::st_linestring(
10       as.matrix(mean.fatal.location[,c("mean.long","mean.lat
11
12 ggplot()+
13   geom_sf(data=mean.fatal.location.path.sf, color="red")+
14   geom_text(data=mean.fatal.location,
15         aes(x=mean.long, y=mean.lat, label=month))
```

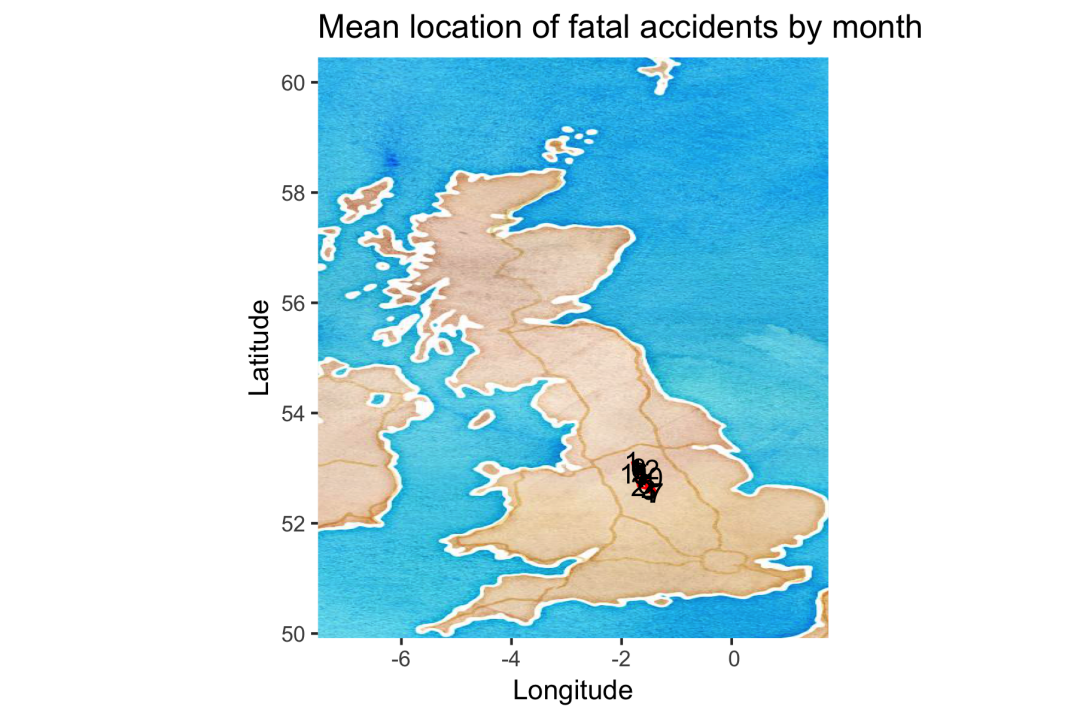
Spatial Data - Exercise 1 II



Spatial Data - Exercise 1 I

```
1  mean.lat <- mean(accidents$Latitude)
2  mean.long<- mean(accidents$Longitude)
3  my.map.dl <- ggmap::get_map(c(left  =min(accidents$Longitude
4                                right =max(accidents$Longitude
5                               maptype="watercolor",  source="
6
7  my.map <- ggmap(my.map.dl)
8
9
10 plot1 <- my.map +
11          ggtitle("Mean location of fatal accidents by month"
12          geom_sf(data=mean.fatal.location.sf, color="red", i
13          geom_text(data=mean.fatal.location,
14              aes(x=mean.long, y=mean.lat, label=month))+
15          ylab("Latitude")+xlab("Longitude")
16 plot1
```

Spatial Data - Exercise 1 II



Spatial Data - Co-ordinate Reference Systems I

Complex - I am by no means an expert!
Geographic co-ordinate system - longitude and latitude.

- Negative longitude is east of prime meridian
- Positive latitude is above equator
- Latitude degree is approximately constant distance; longitude is not
- Surface of the earth is represented by sphere (rare) or ellipsoid
- Ellipsoid also has a datum indicating if optimized for specific points or not optimized etc.

Projected co-ordinate reference system

- Need to map locations from ellipsoid to a flat surface.
- Many different projections, e.g. conic, equal area, etc

The CRS of an object can be retrieved using *sf::st_crs()*

Spatial Data - Co-ordinate Reference Systems II

```
> sf::st_crs(world)
Coordinate Reference System:
  EPSG: 4326
  proj4string: "+proj=longlat +datum=WGS84 +no_defs"

> luxembourg = world[world$name_long == "Luxembourg", ]
> st_area(luxembourg)
2416870483 [m^2]

# careful about setting units
# right number but wrong units
st_area(luxembourg) / 1000000
#> 2414 [m^2]

# right number with right units
units::set_units(st_area(luxembourg), km^2)
```

```
#> 2414 [km^2]
```

Spatial Data - Common CRS I

Geographic CRS

- Most common is WGS84 - most common CRS in world with EPSG 4326.

Projected CRS

- All projected CRS are combinations of equal-area, equidistant, conformal
- Most common projected CRS for "smallish" areas is UTM
  - Earth divided into 60 longitudinal and 20 latitudinal wedges
  - Each point is (Easting, Northing) in meters from intersection of meridian and equator
  - To avoid 0 in Easting, add 500,000

  These are often set by the GIS system used to create the data. Manual setting is possible (see manuals).

Attribute Operations

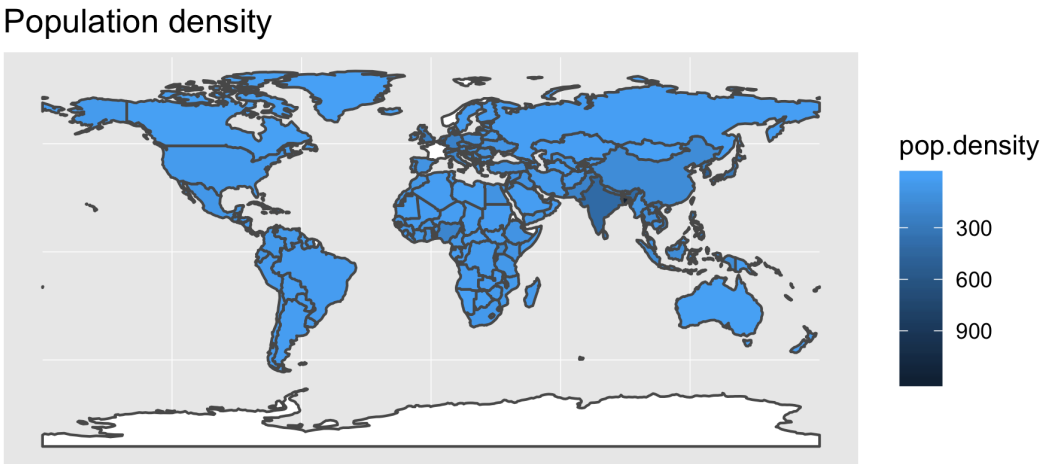
Spatial Data - Attribute Operations I

Because a *sf* object is like a data frame with attributes, the usual attribute operations can be done on it such as merge, transformations etc.
Exercise:

- Make a copy of the world *sf* object
- Create a new attribute - population density
- Plot the population density - reverse the color gradient using *scale_fill_gradient(na.value="white", trans="reverse")*

Spatial Data - Attribute Operations II

```
1  my.world <- world
2  my.world$pop.density <- my.world$pop / my.world$area_km2
3
4  plot1 <- ggplot()+
5     geom_sf(data=my.world, aes(fill=pop.density))+
6     ggtitle("Population density")
7  plot1
```



Spatial Data - Attribute Operations III

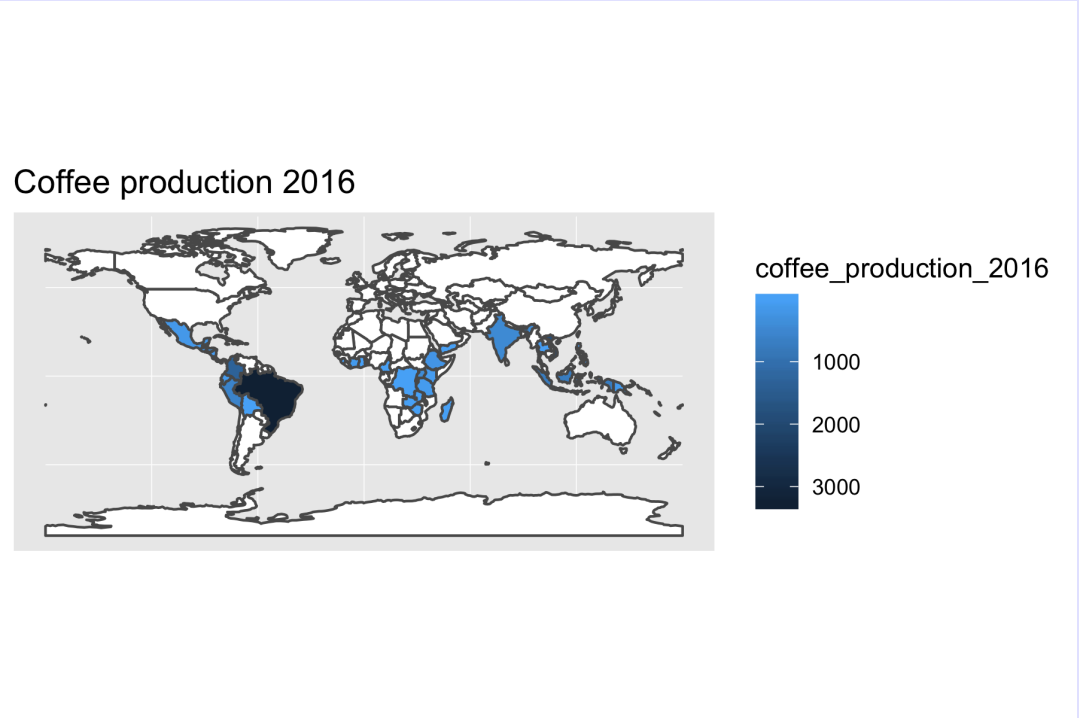
Continuing....

- Get *coffee_data* from *spData* package;
- Check that countries match; correct; merge
- Plot coffee production in 2016

Spatial Data - Attribute Operations IV

```
1  library(spData)
2  names(coffee_data)
3
4  setdiff(my.world$name_long, coffee_data$name_long)
5  setdiff(coffee_data$name_long, my.world$name_long)
6
7  coffee_data$name_long[ coffee_data$name_long=="Congo, Dem. R
8  setdiff(coffee_data$name_long, my.world$name_long)
9
10 my.world2 <- merge(my.world, coffee_data, all.x=TRUE)
11 plot1 <- ggplot()+
12    geom_sf(data=my.world2, aes(fill=coffee_production_2016))
13    ggtitle("Coffee production 2016")
14 plot1
```

Spatial Data - Attribute Operations V



NOTE: See what happens if you forget it all.x=TRUE

Geometry Operations

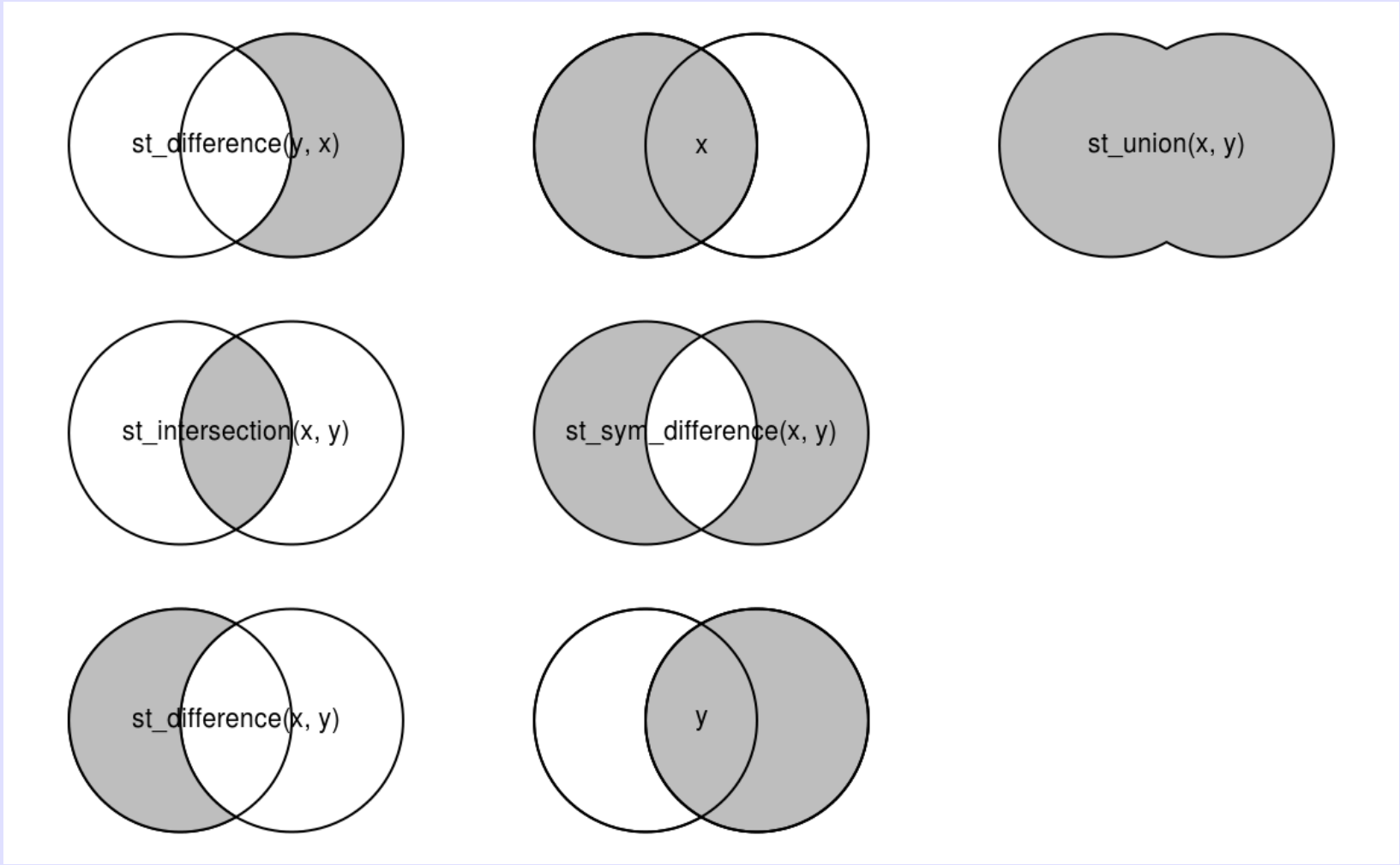
Spatial - Geometry Operations I

Complete set of geometry operations.

- *sf::st_simplify()* - reduces the complexity of a line or polygon, i.e. smoothes
- *sf::st_centroid()* - computes the geographic centroid of an object
- *sf::st_buffer()* - computes a buffer around an object

Spatial - Geometry Operations II

Different intersections:



Spatial - Geometry Operations III

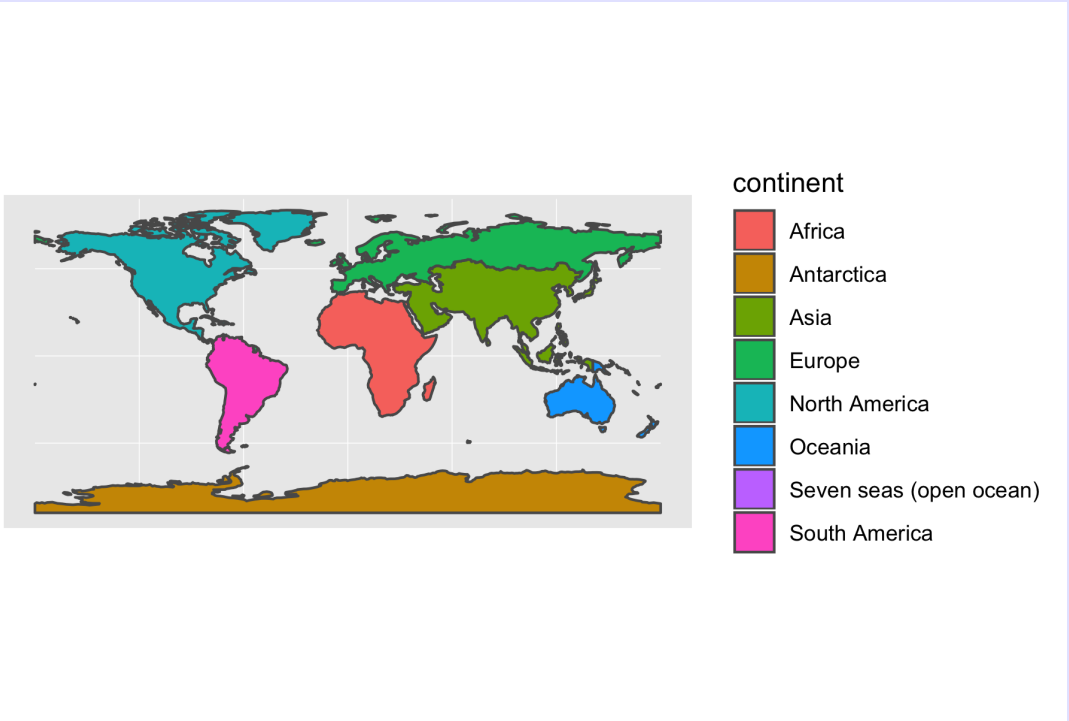
- *sf::st_union()* - combined two (or more objects) into a new combined object

Exercise: create a map of population density by continents from the world map

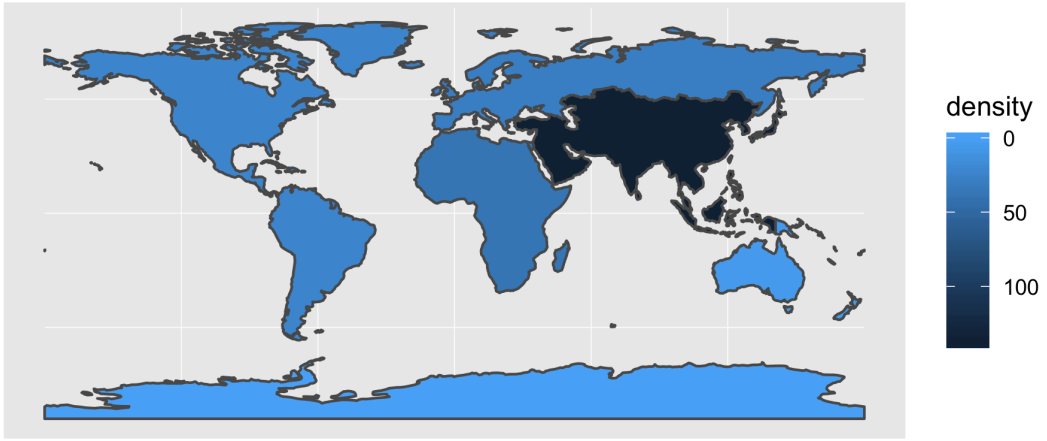
Spatial - Geometry Operations IV

```
cont <-
  world %>%
    group_by(continent) %>%
      summarize(
          total.pop =sum(pop,na.rm=TRUE),
          total.area=sum(area_km2, na.rm=TRUE),
          density = total.pop / total.area)
str(cont)
cont

ggplot()+
  geom_sf(data=cont, aes(fill=continent))
  scale_fill_gradient(trans="reverse", na.value="white")

ggplot()+
  geom_sf(data=cont, aes(fill=density))+
  scale_fill_gradient(trans="reverse", na.value="white")
```

Spatial - Geometry Operations V



Spatial - Geometry Operations VI



Reading/Writing Geographic data

UGH!!!

Reading Geographic data

- RTFM! Extremely complex.
- At least 200 vector and raster formats!
  - *shapefiles* from ESRI is a common interchange format, but not the best
  - *rgdal* - unified structure for different format
  - *st_read()* in *sf* covers most of *rgdal*

Writing Geographic data

- *rgdal* - unified structure for different format
- *st_read()* and *st_write()* in *sf* covers most of *rgdal*

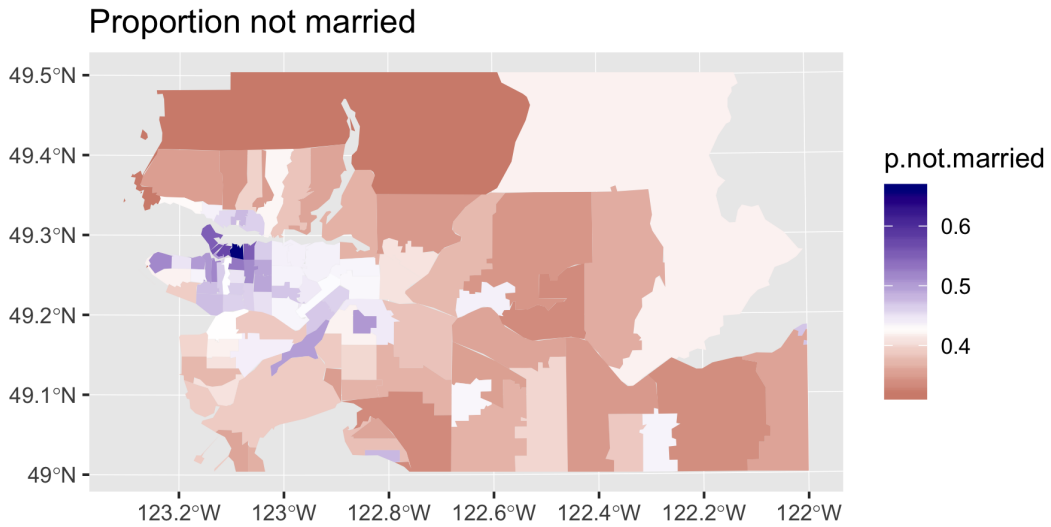
Refer to `https://geocompr.robinlovelace.net/adv-map.html#mapping-applications` for introduction to making interactive maps.

Spatial Data Exercise

Looking at results of 2016 census of Canada

Proportion of not married by FSA I

We wish to produce:



Proportion of not married by FSA I

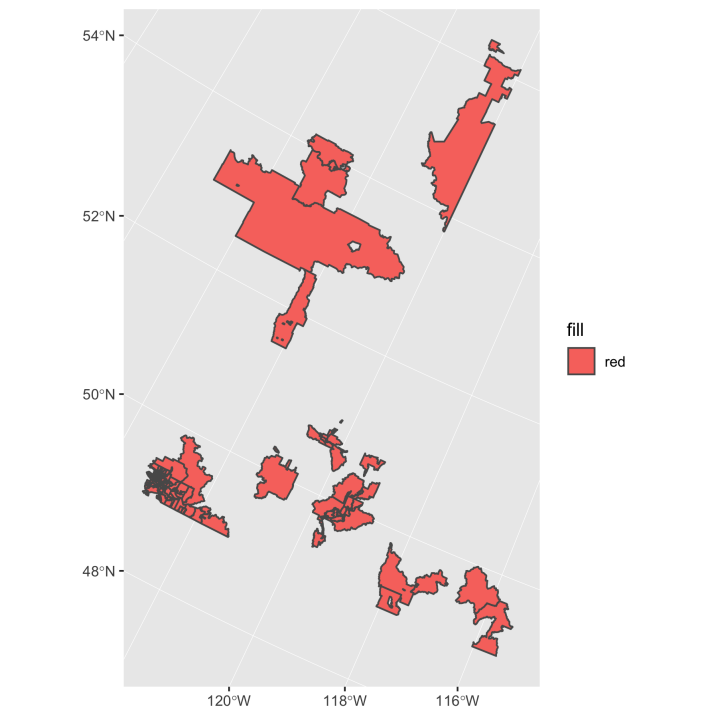
Complete the following steps:

- Read in FSA for all of Canada. Look in *sampledata* for 2016-census directory.
- Need to point to **.shp* file with *sf::st_read()*
- Select only those FSA in lower part of BC. First two character are "V3", "V4",...."V7"
- Simplify the FSA boundaries to reduce file size and rendering time. Use *dTolerance=200*.

Proportion of not married by FSA II

```
fsa <- sf::st_read(file.path(.....,
          "lfsa000b16a_e.shp"), stringsAsFactors=FALSE)
head(fsa)

# Extract only bc
xtabs(~PRNAME, data=fsa, exclude=NULL, na.action=na.pass)
fsa <- fsa[ substr(fsa$CFSAUID,1,2)
       %in% c("V3","V4","V5","V6","V7"),]

# simplfy the boundaries
fsa <- sf::st_simplify(fsa, dTolerance=200)

plot1 <- ggplot()+
  geom_sf(data=fsa, aes(fill=NA))
plot1
```

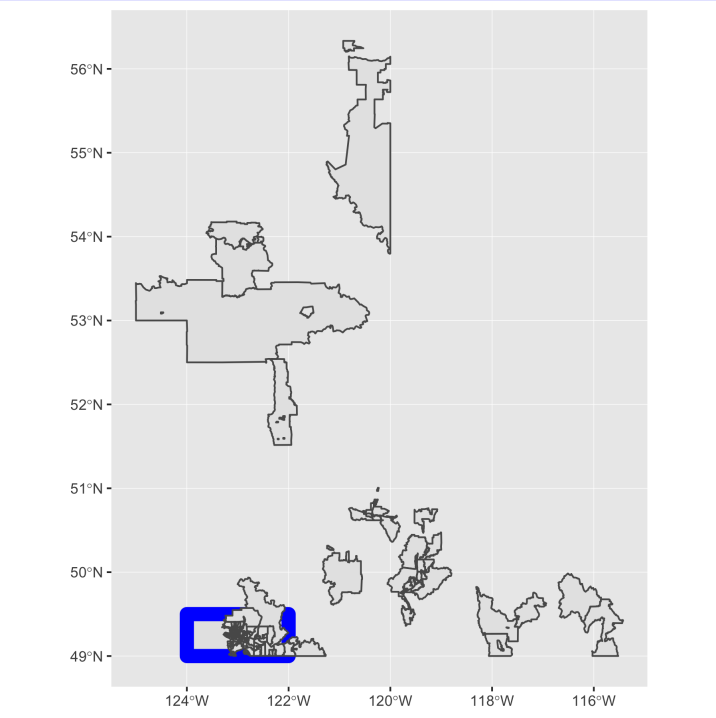
Proportion of not married by FSA III



There are a few oddly names FSA that are not near the lower mainland??

Proportion of not married by FSA IV

- Create a bound box for FSA in lower mainland.
  my.bbox <- data.frame(long=c(-122, -122, -124, -124, -122), lat =c(49, 50, 50, 49, 49))
- Add the box on the plot.



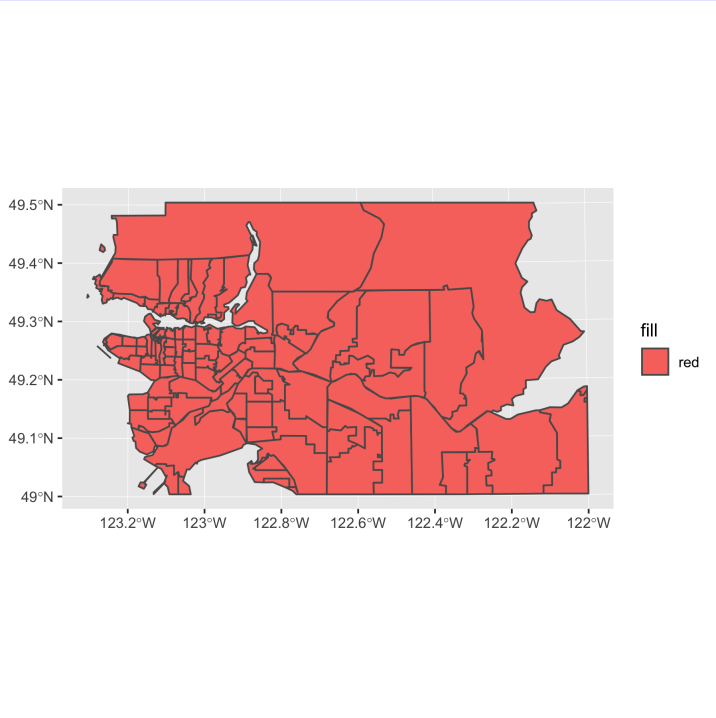
Proportion of not married by FSA V

```
my.bbox <- data.frame( long=c(-122, -122, -124, -124, -122)
                       lat =c(  49,   50,   50,   49,   49))
                  my.bbox.sf <- st_sfc(st_polygon(list(as.m
plot2 <- ggplot()+
  geom_sf(data=fsa, aes(fill="red"))+
  geom_sf(data=my.bbox.sf, color="blue", size=4, aes(fill=NU
plot2
```

Proportion of not married by FSA VI

- Transform the current FSA and bounding box to UTM *crs="+proj=utm +zone=10 ellps=WGS84"*
- Only keep those portions of FSA within the bounding box.

You should end up with something like:



Proportion of not married by FSA VII

```
my.bbox.sf <- sf::st_transform(my.bbox.sf, crs="+proj=utm +z
fsa        <- sf::st_transform(fsa,        crs="+proj=utm +z

fsa <- sf::st_intersection(fsa, my.bbox.sf)
plot3 <- ggplot()+
  geom_sf(data=fsa,aes(fill=NULL))
plot3
```

Proportion of not married by FSA VIII

- Read in the census data on marital status.
- Compute proportion of not current married nor common law. Group "9" / Group "1" totals.
- Only keep those data points in the selected FSA

You should get something like:

```
> head(p.not.married)
  GEO_NAME p.not.married
1 V3A              0.436
2 V3B              0.409
3 V3C              0.409
...
```

Proportion of not married by FSA IX

```
m.status <- read.csv(.....,
    "98-400-X2016039_English_CSV_data.csv"), header=TRUE, as
m.status <- m.status[ m.status$GEO_NAME %in% fsa$CFSAUID,]
m.status <- m.status[ m.status$DIM..Sex..3. == 'Total - Sex
names(m.status)

# get the proportion who are "Not Married and not common la
library(dplyr)
p.not.married <-
 m.status %>%
  group_by(GEO_NAME) %>%
   do(
     (function(x){
        #browser()
        p.not.married =
           x$Dim..Age..16...Member.ID...1...Total...Age[x$Me
           x$Dim..Age..16...Member.ID...1...Total...Age[x$Me
        data.frame(p.not.married)
```

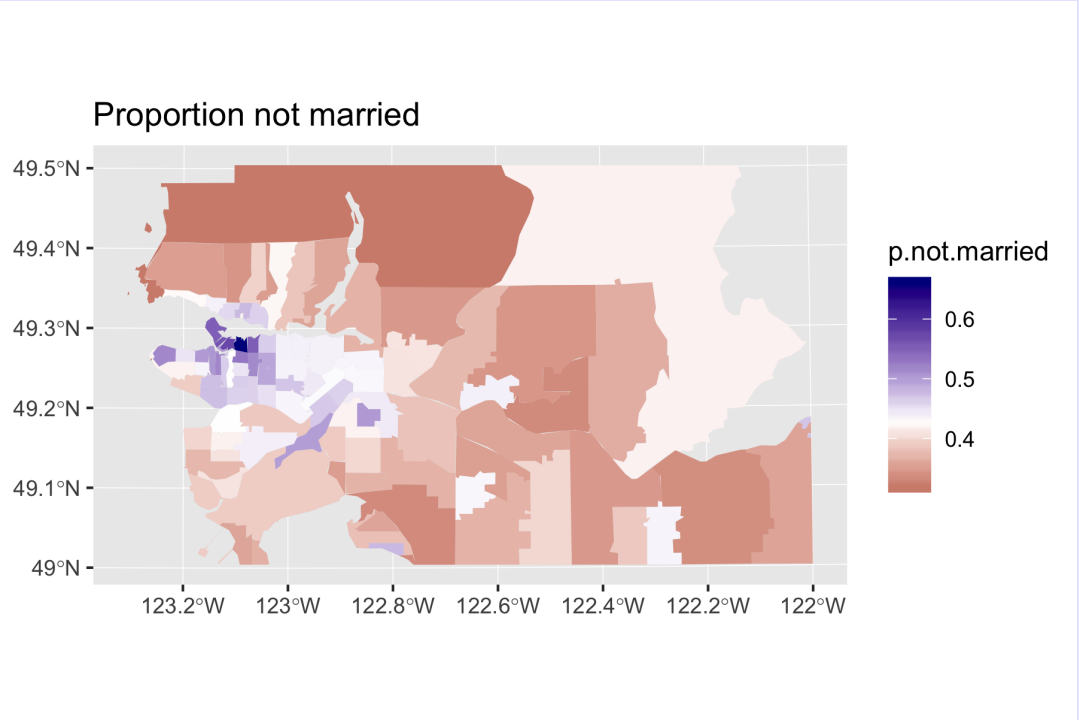
Proportion of not married by FSA X

```
19          })(.)
20      )
21  head(p.not.married)
```

Proportion of not married by FSA XI

- Merge with the FSA data
- Plot with color heatmap
- Use *scale_fill_gradient2()* to fit a divergent scale around the midpoint.

You should get something like:



Proportion of not married by FSA XII

```
setdiff(fsa$CFSAUID, p.not.married$GEO_NAME)
setdiff(p.not.married$GEO_NAME, fsa$CFSAUID)

fsa <- merge(fsa, p.not.married, by.x="CFSAUID", by.y="GEO_N

final.plot <- ggplot()+
  ggtitle("Proportion not married")+
  geom_sf(data=fsa,aes(fill=p.not.married), color=NA)+
  scale_fill_gradient2(low='darkred', mid="white", high='dar
final.plot
```

Proportion of not married by FSA- *leaflet* I

Complete the following steps to generate an interactive *leaflet* map

```
1  library(leaflet)
2  ### Create five colors for fill
3  mypal <- colorQuantile(palette = "RdYlBu", domain = fsa_wgs8
4
5  # we need to transform the FSA inot WGS84
6  fsa2 <- st_transform(fsa, '+proj=longlat +datum=WGS84')
7
8  m <- fsa2 %>%
9       leaflet() %>%
10       addTiles() %>%
11       addPolygons(data = fsa2,
12             stroke = FALSE, smoothFactor = 0.2, fillOpacity
13             fillColor = ~mypal(fsa2$p.not.married)) %>%
14       addLegend(position = "bottomright", pal = mypal, fsa_w
15           title = "P(not married)",
16           opacity = 1)
17 m  # display the map
```

Proportion of not married by FSA- *leaflet* II