BIOM4051 Tutorial 2: Advanced Vector Operations

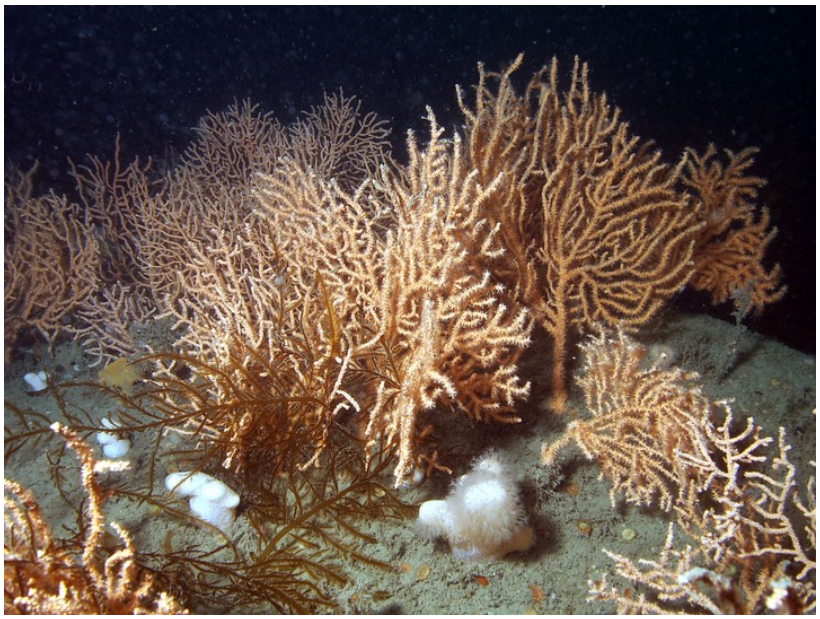


Figure 1: The pink sea fan (Eunicella verrucosa)

Introduction

In this tutorial we are going to build on basic vector handling skills that you learned in Tutorial 1 and introduce a range of more advanced operations, including **vector geoprocessing, spatial queries and joins**.

The datasets we will be using relate to the conservation of the Pink Sea Fan (*Eunicella verrucosa*) in Falmouth Bay. The Pink Sea Fan is a cold water, gorgonian soft coral found in the NE Atlantic and western Mediterranean, with several strongholds in the south west of England. It is a slow growing species and colonies are slow to recover/recolonize once damaged, making it highly susceptible to bottom-towed fishing gears such as trawls. Its presence can therefore be regarded as a reliable indicator of relatively undisturbed benthic habitats.

The pink sea fan is listed as 'vulnerable' on the IUCN Red List and is protected in the UK under the Wildlife and Countryside Act 1981 and the NERC Act 2006. They are listed as 'features of conservation importance' (FOCI) for creation of Marine Conservation Zones (MCZ), several of which have been designated along the SW coast of England. Seabed habitats associated with pink sea fans may also be protected in Special Areas of Conservation (SACs) designated under the EU Habitats Directive.

In this tutorial we are going to map the distribution of pink sea fans in Falmouth Bay and evaluate how effective existing MPAs are in protecting the local population and its habitats. The tutorial replicates elements of a regional analysis by Pikesley et al. (2016) which you can refer to for further information.

Intended learning outcomes

By the end of this tutorial you should be able to:

1. Perform a range of common vector geoprocessing operations using R (including crops, intersections and differences).
2. Understand how spatial queries work in R and use topological relations to filter and count vector features.
3. Convert spatial vectors between different geometry types.
4. Dissolve, or union, feature geometries by attribute.
5. Perform basic spatial joins to link attributes from different vector datasets.

Datasets

This case study uses spatial datasets obtained entirely from openly accessible online sources. In the Practical 1 data folder you will find the following files:

1. **MCZs.gpkg** Boundaries of MCZs in England downloaded from here.
2. **SACs.gpkg** Boundaries of SACs in England downloaded from here.
3. **pink_sea_fans.csv** Pink sea fan occurrence records in the UK downloaded from the National Biodiversity Network. These consist primarily of diver-reported sightings from the Seasearch citizen science programme (described in Pikesley et al. 2016) but supplemented with data from other sources.
4. **uk_seamap.shp** A benthic habitat map for the UK downloaded from UKSeaMap 2018 and pre-cropped to our study area to reduce file size.
5. **falmouth_bay.shp** A land polygon of the Falmouth Bay area (used in Tutorial 1).
6. **uk.shp** A shapefile of the UK obtained from the Office for National Statistics Geoportal.

1. Data preparation

As always, start by setting the working directory and loading in any packages that you will need. As in Tutorial 1, we will be using `sf` and `tidyverse`. We will also introduce the package `colorspace` which is one of the best R packages for generating custom colour palettes:

```
library(sf)
library(tidyverse)
library(colorspace)
```

Let's begin by inspecting the `pink_sea_fans.csv` dataset. The file contains latitudes and longitudes of pink sea fan observations in XY format so we read in the file using `read.csv` and convert it into an `sf` object using `st_as_sf`. The coordinate reference system (crs) of these data is WGS84 latitude and longitudes, which has EPSG code 4326:

```
fans = read.csv('data/pink_sea_fans.csv')
fans = st_as_sf(fans,coords = c('Longitude','Latitude'), crs = 4326)
```

The sightings dataset we have downloaded contains records going back to 1960. You can confirm that be typing `range(fans$Start.date.year)` in the console. Given that the distribution of the pink sea fan may have changed over time, we will limit our analysis to only observations collected since 2000 using `subset`. We can also use the 'select' argument in `subset` to drop a lot of attribute columns that we aren't interested in and just keep the `Common.name` column:

```
fans = subset(fans, Start.date.year >= 2000, select = 'Common.name')
```

The processing steps above on out `fans` data can be simplified using the `tidyverse` 'pipe operator', `%>%`, to pipe the output of each step into the first input of the next:

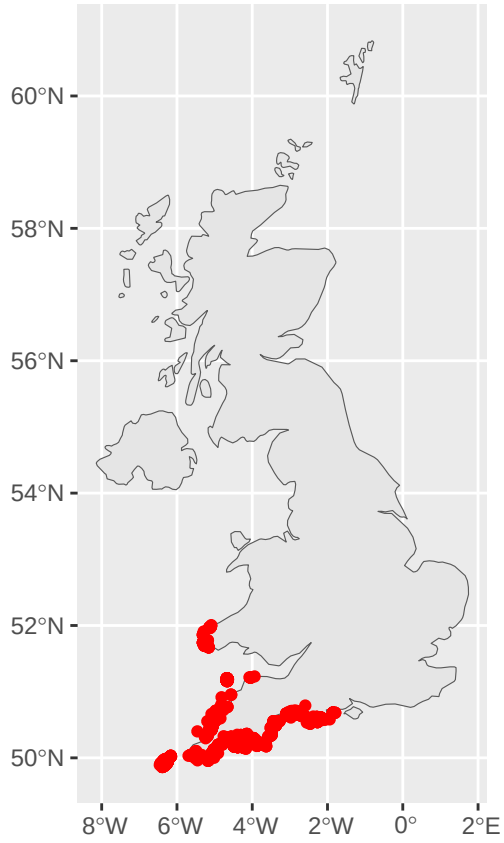
```
fans = read.csv('data/pink_sea_fans.csv') %>%
       st_as_sf(coords = c('Longitude','Latitude'), crs = 4326) %>%
       subset(Start.date.year >= 2000, select = 'Common.name')
```

The result is the same, but the code is more compact and the steps that have been applied are easier to follow. Run this a few times to check you understand what is happening.

This leaves us 3988 records since the year 2000. Let's load in our UK map from the Tutorial 2 data folder and plot the data to inspect it using `ggplot`:

```
uk = st_read('data/uk.shp')

ggplot() +
  geom_sf(data = uk) +
  geom_sf(data = fans, colour = 'red')
```



You can see that the UK distribution of this species is centred on SW England and a small area around the coast of Permbrokeshire in west Wales.

2. Vector geometry operations using `st_crop()`, `st_difference()`, `st_intersection()` and others

Now let's look a bit closer to home at the distribution of pink sea fans in Falmouth Bay. You will need to read in your `falmouth_bay.gpkg` file of our Falmouth Bay study area that we used in Tutorial 1.

```
fal.bay = st_read('data/falmouth_bay.shp')
```

To extract only the observations in Falmouth Bay we need to crop the `fans` dataset to include only the sightings in the defined area we are interested in. The function for cropping `sf` objects to a defined rectangle is called `st_crop()`. You can either give `st_crop()` the bounding coordinates of the rectangle or you can give it another `sf` object and the function will use the bounding box surrounding it to crop by. Here, we will find all of the records that fall within in the box surrounding our `fal.bay` polygon (you will need to transform `fans` to the same CRS as `fal.bay` or you will get an error):

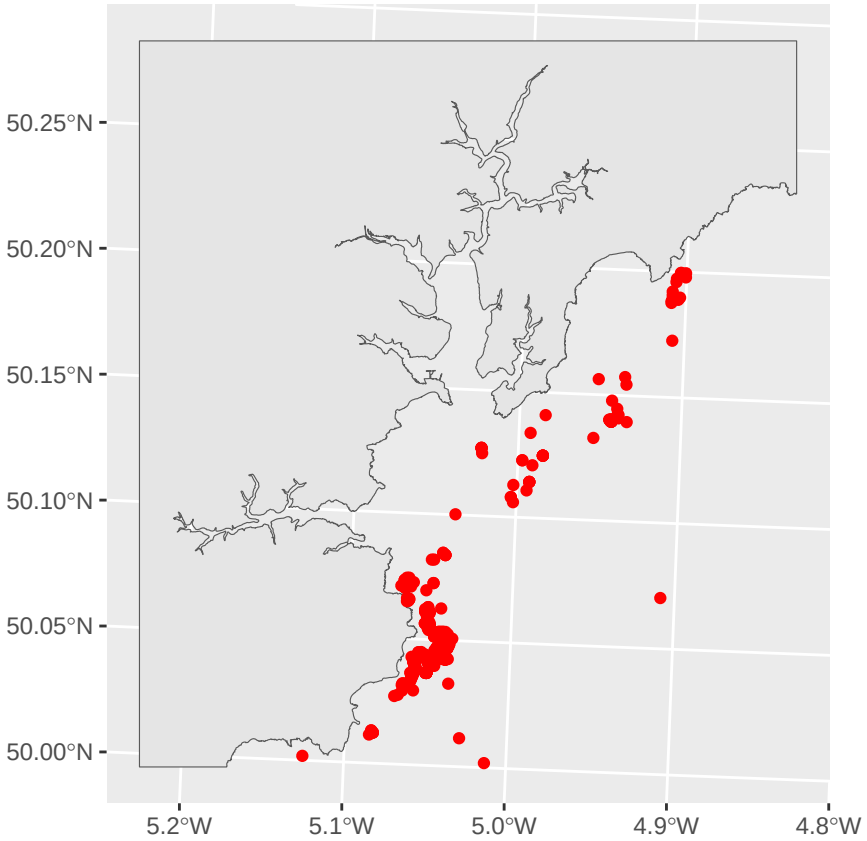
```
fans = st_transform(fans, crs = 27700)

fb.fans = st_crop(fans,fal.bay)
```

Now let's plot our Falmouth Bay sightings using `ggplot`, creating a new plot called `fan.plot`:

```
fan.plot =
  ggplot() +
  geom_sf(data=fal.bay) +
  geom_sf(data=fb.fans, colour='red')

fan.plot
```

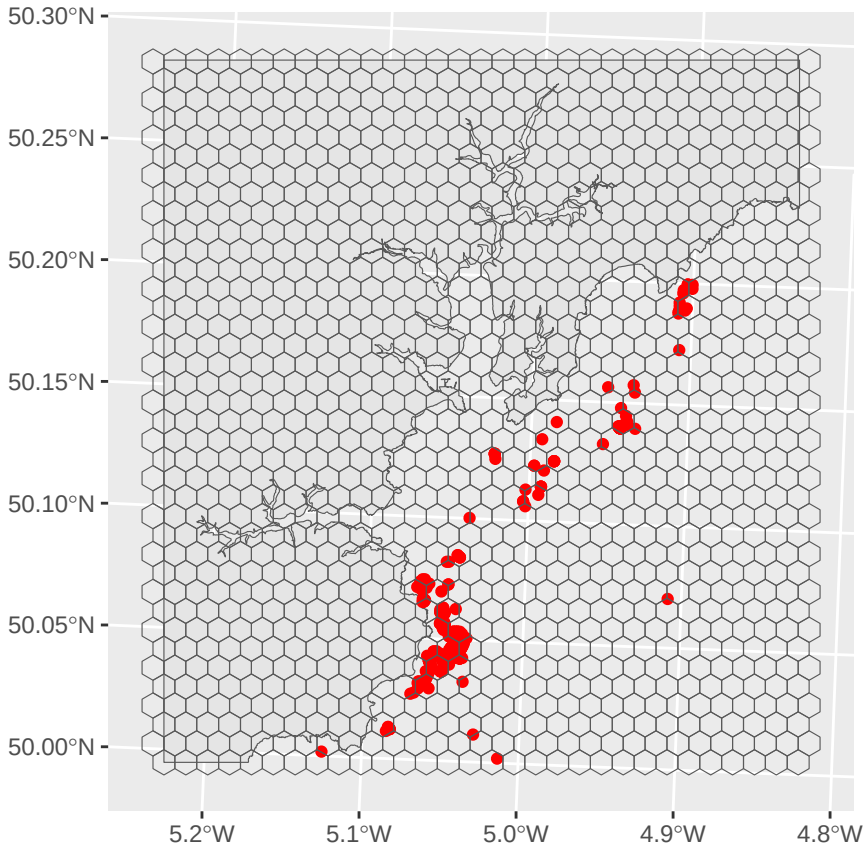


You can see Pink Sea Fan sightings appear to be distributed in a defined belt across the mouth of the bay, with a pronounced cluster at a latitude of about 50.05 N which is the Mannacles Rocks: a popular dive site.

Next, let's measure the "Area of Occupancy" (AOO) of pink sea fans in Falmouth Bay. AOO is generally defined as the total area of regularly sized grid cells that are occupied by a species, so we begin by creating an overlay grid of 1 x 1 km cells using the `st_make_grid()` function. By default `st_make_grid()` produces standard square grid cells, but we can use hexagonal cells instead by setting `square=F`:

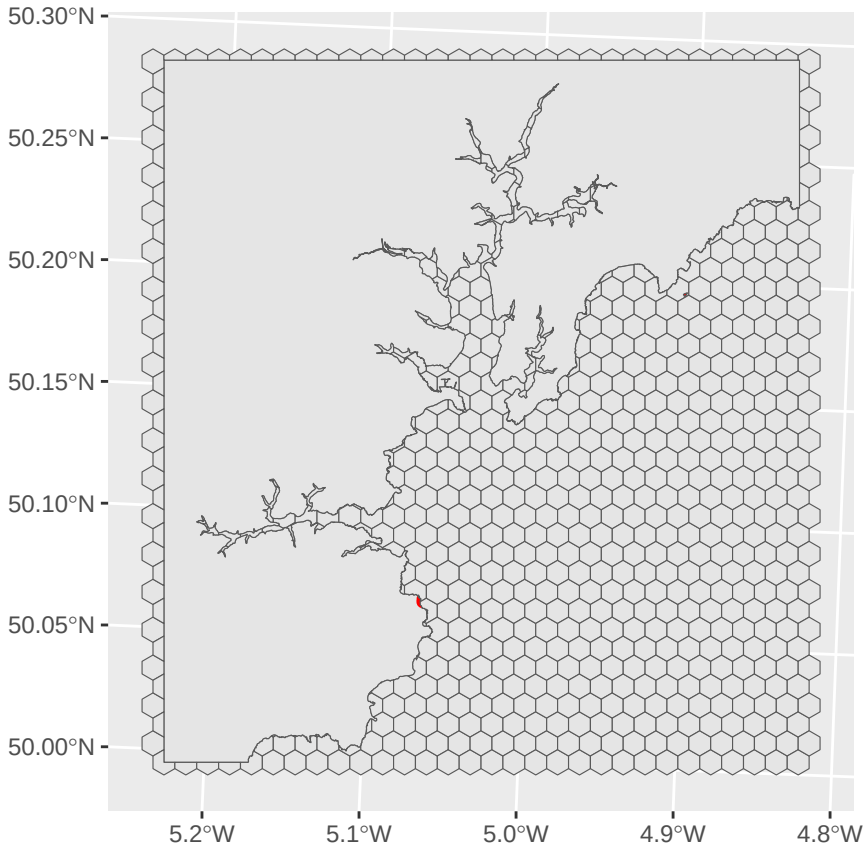
```
grid = st_make_grid(fal.bay,cellsize=1000,square = F)

# Set fill to transparent or we won't be able to see data!
fan.plot + geom_sf(data = grid,fill = 'transparent')
```



A lot of our grid overlaps land, which isn't great. We want to remove the bits of `grid` that overlap our `fal.bay` land polygon. Operations like this that modify one geometry based on another are often called "geometric operations". There is a nice figure illustrating the different functions that are available in `sf` and what they each do in Geocomputation with R. In this case we want `st_difference`

```
fan.plot + geom_sf(data = st_difference(grid, fal.bay))
```



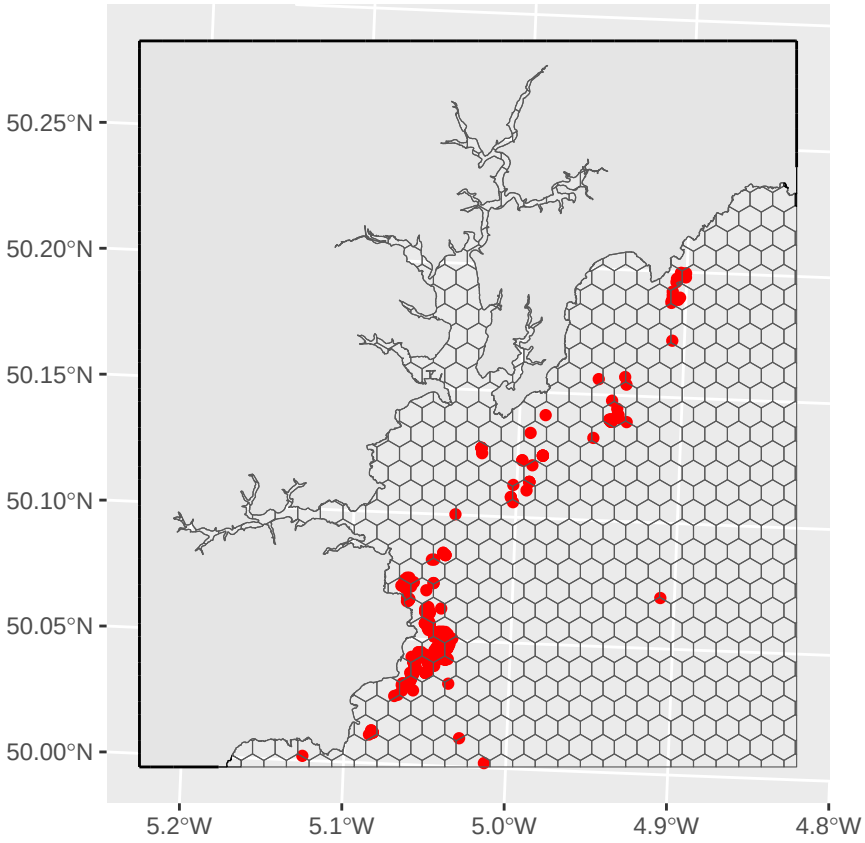
The opposite is called `st_intersection` which would retain the bit of `grid` that intersects the land polygon:

```
fan.plot + geom_sf(data = st_intersection(grid, fal.bay))
```

We might also want to crop our grid to the bounding box of our study area using `st_crop` to keep things neat:

```
grid = st_difference(grid, fal.bay) %>%
       st_crop(fal.bay)

fan.plot + geom_sf(data = grid,fill = 'transparent')
```



This looks good. There is one final processing step we need to do. If we print a summary of `grid` you will see that this is currently a 'Geometry Set' object which is basically just a geometry column with no attribute table to populate. In order to combine this with our pink sea fans points we need to convert it to a normal `sf` object, or spatial dataframe, using `st_as_sf`. So, our final processing pipe looks like this:

```
grid = st_make_grid(fal.bay,cellsize=1000,square = F) %>%
       st_difference(fal.bay) %>%
       st_crop(fal.bay) %>%
       st_as_sf()
```

Next we want to determine how many of these cells contain pink sea fan observations. do this we need need to introduce another class of functions called 'topological relations' or spatial queries

3. Spatial queries using `st_intersects()` and other topological relations

In broad terms, spatial queries involve subsetting one set of geometries based on their spatial relationship with another. In `sf`, spatial queries are built around a set of functions called 'topological relations' which take two geometries and test whether a particular condition is met describing the geometric relation of the first to the second. For example, `st_intersects(x,y)` will return TRUE if each feature in x intersects (i.e. shares any space with) each feature in y, or FALSE if it doesn't. We can use this type of function to spatially filter datasets, or to count features for which a condition is TRUE. Let's use `st_intersects` to check whether each cell in our grid intersects (contains) pink sea fan sightings (we specify `sparse = F` here to see the full output):

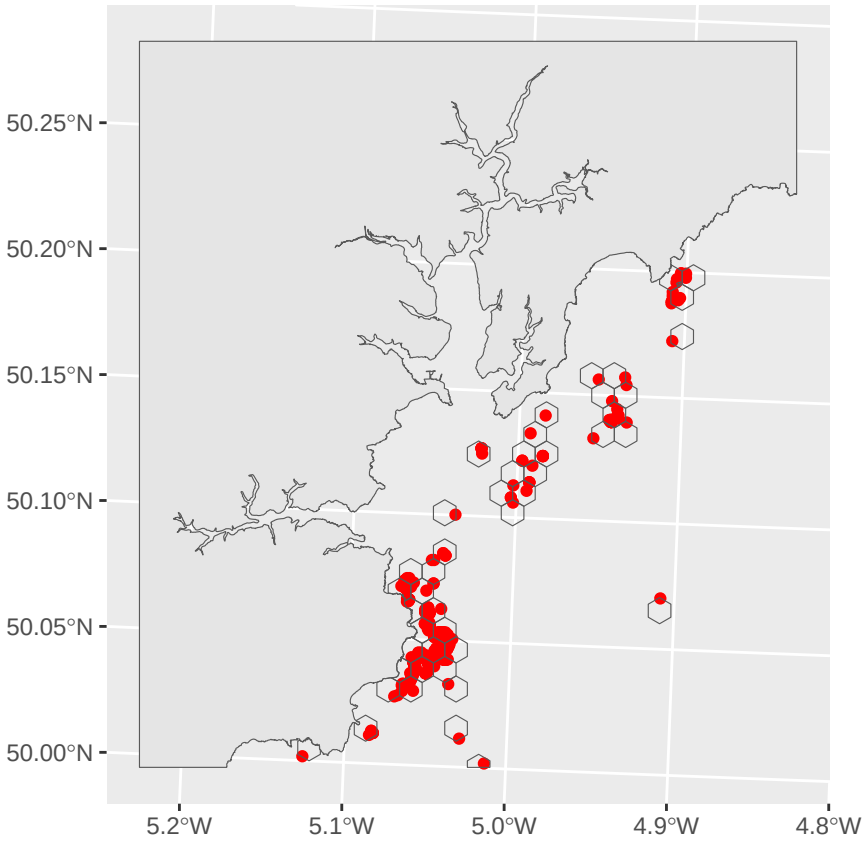
```
# We could use st_contains for the same purpose
st_intersects(grid,fb.fans,sparse=F)
```

This returns a matrix of TRUE/FALSE that tells you whether each grid cell in the first layer (rows) intersects each sighting point in the second layer (columns). What if we want to filter only the cells that contain at least one sighting? There are two common notations for doing this, either using `st_filter()` or using normal square bracket notation. Both of these are equivalent:

```
st_filter(grid,fb.fans)
grid[fb.fans,]
```

If we print this object you can see that 45 cells have been selected, and we can plot this to confirm:

```
grid = grid[fb.fans,]
fan.plot + geom_sf(data = grid, fill = 'transparent')
```



By default, spatial queries using either `st_filter` or square bracket notation uses `st_intersects` as the topological relation, which is a catch-all for saying two geometries share any part of space (e.g. they overlap, touch, contain). However, there are a range of alternative relations we can use. Type `?st_intersects` into your R console and see Figure 4.2 in Geocomputation with R for a list of options and what they do.

Another one that may come in useful in our case is `st_is_within_distance`. Our dataset contains some dubious records located a long way from shore that we might not trust (remembering that these observations are citizen science data and coordinates might have been entered wrong). Let's use the `st_is_within_distance` relation to filter our `fans` dataset and retain only observations that are within 4km of shore. Again, both of these syntaxes are equivalent:

```
st_filter(fb.fans,fal.bay,dist=4000,.predicate = st_is_within_distance)

fb.fans = fb.fans[fal.bay,,op = st_is_within_distance, dist=4000]
```

If we plot the result, you can see that records > 4km shore have been removed.

```
fan.plot = ggplot() +
           geom_sf(data = fal.bay) +
           geom_sf(data = fb.fans,colour='red')
```

We can now recreate our `fan.plot` and our grid of occupied cells minus the dubious records

```
fan.plot =
  ggplot() +
  geom_sf(data=fal.bay) +
  geom_sf(data=fb.fans, colour='red')

grid = grid[fb.fans,]
fan.plot + geom_sf(data = grid, fill = 'transparent')
```

CHALLENGE: *What is the area of occupancy of pink sea fans in Falmouth Bay? Use* `st_area` *to work it out.*

So far we have generated an 'occupancy grid' with at least one sighting per cell. However, some cells obviously contain a lot more observations than others. How do we count the number of observations in each cell in order to better map the distribution? This type of operation also uses a spatial query, but this time rather than finding rows in our matrix which contain at least one TRUE, we want to count how many TRUES occur in each row. Fortunately the summary for `st_intersects` (obtained by leaving `sparse=FALSE`) makes this easy to calculate:

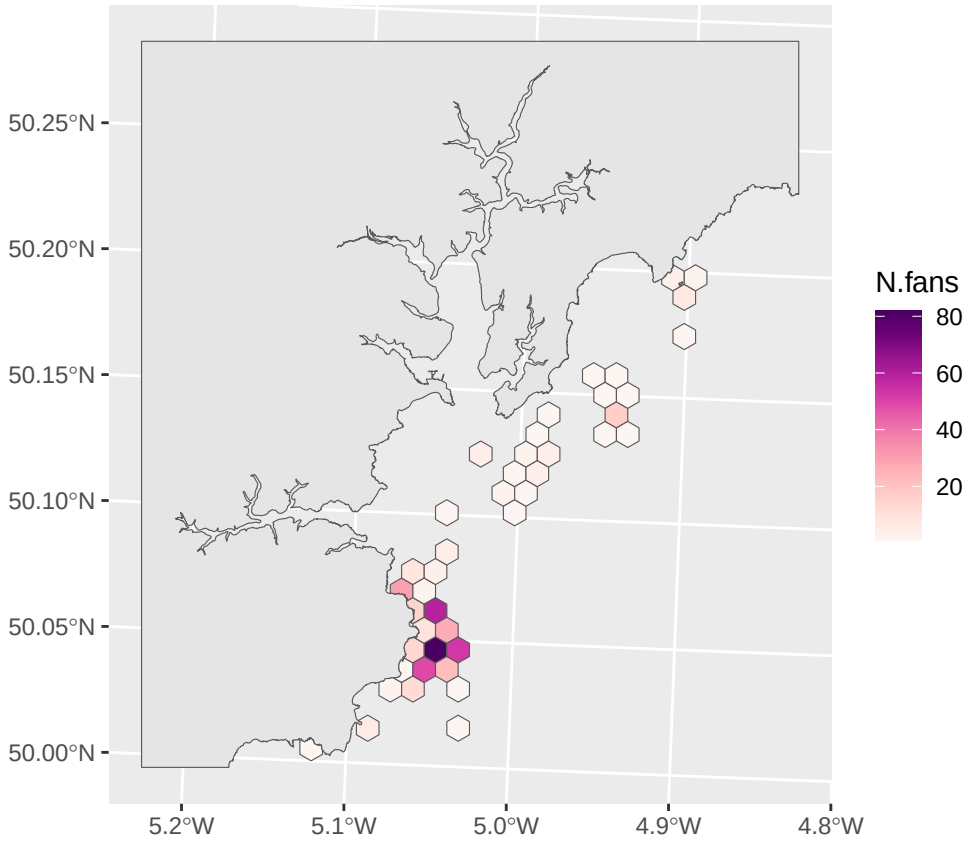
```
st_intersects(grid,fb.fans)
```

This time the output is a list which contains the IDs of the sightings points that intersect each grid cell. All we need to do is count how many IDs there are in each element of the list using the `lengths` function, and pass the result to a new column in our `grid` dataset.

```
grid$N.fans = lengths(st_intersects(grid,fb.fans))
```

We can then plot our data using a custom colour palette from package `colorspace`. You can view the palettes that are available by typing `hcl_palettes(plot=T)` into your console. We will use the red-purple 'RdPu' palette. As with all custom ggplot scales, the functions for adding colorspace palettes to a plot always being with 'scale_', followed by the aesthetic you want to modify, in our case 'fill'. We then need to indicate the type of data we have (continuous or discrete) and the type of colorspace palette (e.g. sequential, diverging). So in our case we want `scale_fill_continuous_sequential`:

```
ggplot() +
  geom_sf(data = fal.bay) +
  geom_sf(data = grid, aes(fill = N.fans)) +
  scale_fill_continuous_sequential(palette = 'RdPu')
```

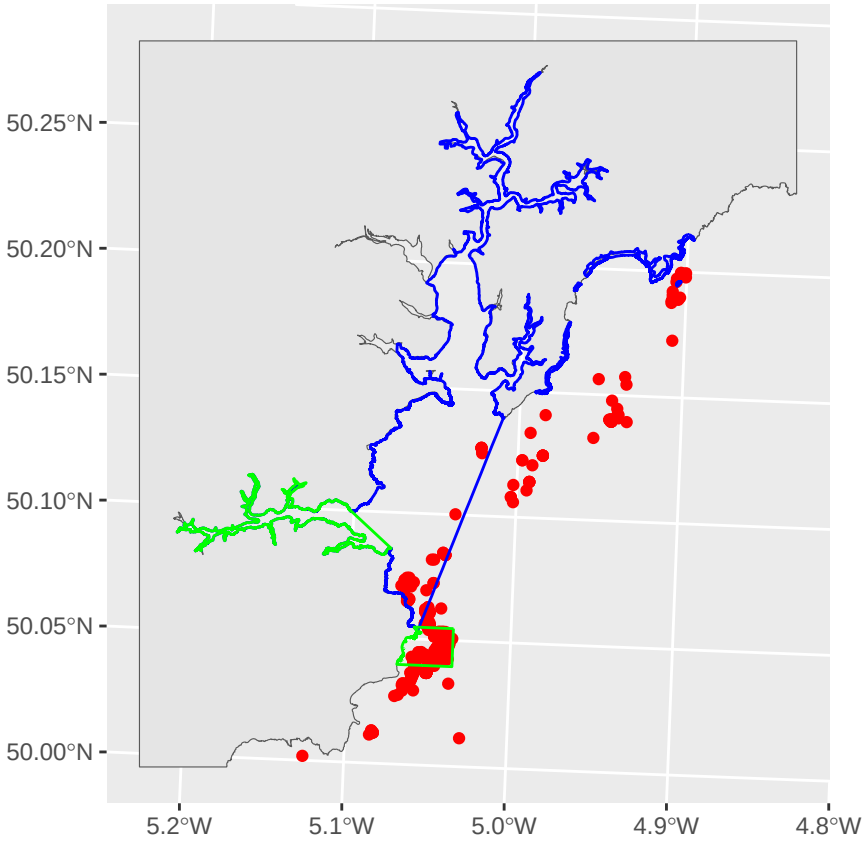


CHALLENGE: *Because we clipped our grid to land, the cells are not all of equal area. See if you can calculate and plot the density of pink sea fan sightings in each cell in number/km^2 to more accurately represent their distribution.*

Now that we have cleaned and mapped our data, let's come back to our main research question: *"How effective are existing MPAs in Falmouth Bay at protecting pink seas fans in terms of their spatial coverage?"* First, let's read in our MPA boundaries from the Tutorial 2 data folder and plot them on our map:

```
mcz = st_read('data/MCZs.gpkg')
sac = st_read('data/SACs.gpkg')

fan.plot +
  geom_sf(data = sac, colour = 'blue') +
  geom_sf(data = mcz, colour = 'green')
```



Counting how many fan observations occur inside each of our MPAs is a simple spatial subsetting operation like we have just performed above. However, if we try and filter our `fb.fans` dataset to find observations that intersect `mcz` or `sac` we get zero features:

```
fb.fans[mcz,]
fb.fans[sac,]
```

That is because the MPA boundaries we just imported are line features and none of the observation points exactly intersect those lines. If we want to find points within MPAs, we need to first convert them to polygons.

4. Geometry type conversions using `st_cast()`

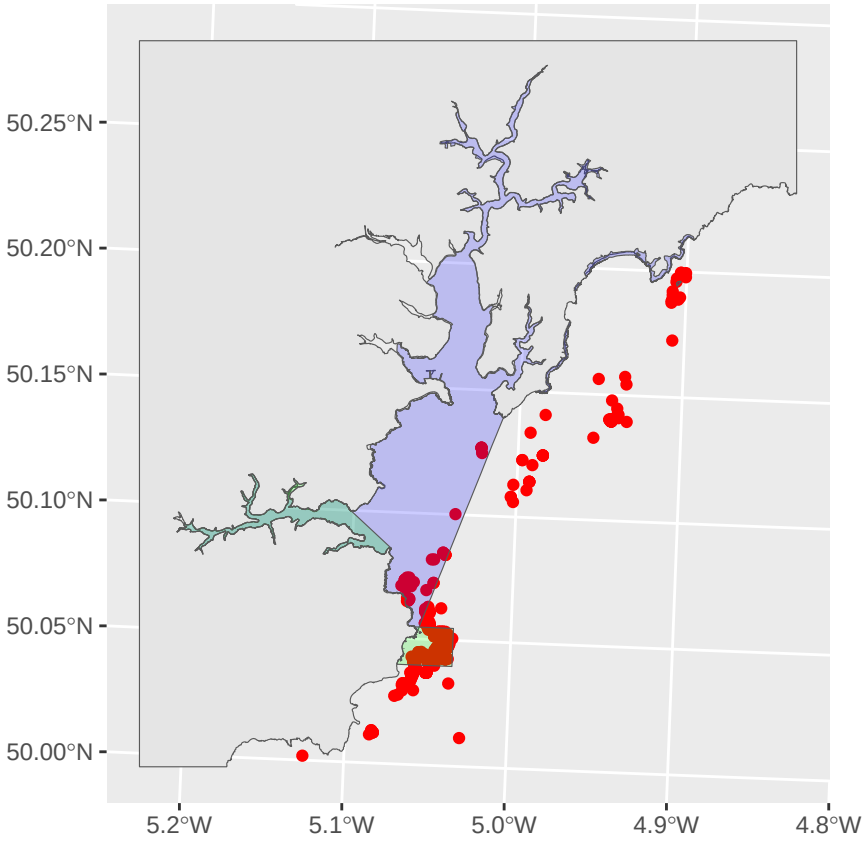
In `sf`, conversion between geometry types is done using a function called `st_cast`. There is a summary of the different geometry types that are supported in Figure 2.2 of Geocomputation with R. In our case, the MPA boundaries are 'MULTILINESTRING' geometries (i.e. single features are made up of multiple lines), so we need to convert them to 'MULTIPOLYGON' geometries.

```
mcz = st_cast(mcz,'MULTIPOLYGON')
sac = st_cast(sac,'MULTIPOLYGON')

fan.plot =
fan.plot +
  geom_sf(data = sac, fill = 'blue',alpha=.2) +
```

```
  geom_sf(data = mcz, fill = 'green', alpha=.2)

fan.plot
```



If we subset fans now, this works and we get 220 sightings within MCZs and 73 within the SAC.

```
fb.fans[mcz,]
fb.fans[sac,]
```

Note this doesn't tell us exactly how many are located inside each MCZ. If we want to do that then we need to count points in polygons like we did earlier `lengths(st_intersects(mcz,fb.fans))`, which tells us that all 220 were in the Manacles Rocks MCZ and none in the Helford Estuary MCZ.

CHALLENGE: With the help of ***nrow****, calculate the proportions of pink sea fans protected within each MPA designation*

Currently, about two thirds of pink sea fan sightings reported in Falmouth Bay occur within MPAs, which seems quite good. However, there are currently some significant limitations with the approach we have taken. In particular, our sightings data are likely to be heavily biased towards a few highly frequented areas, including popular dive sites like the Manacles MCZ. As an alternative approach, we could look at how well existing MPAs protecting suitable habitat for sea fans in Falmouth Bay, including areas that have not yet been surveyed. To do that, we need to understand a bit more about the habitat preferences of the Pink Sea Fan.

5. Dissolving geometries using `st_union`

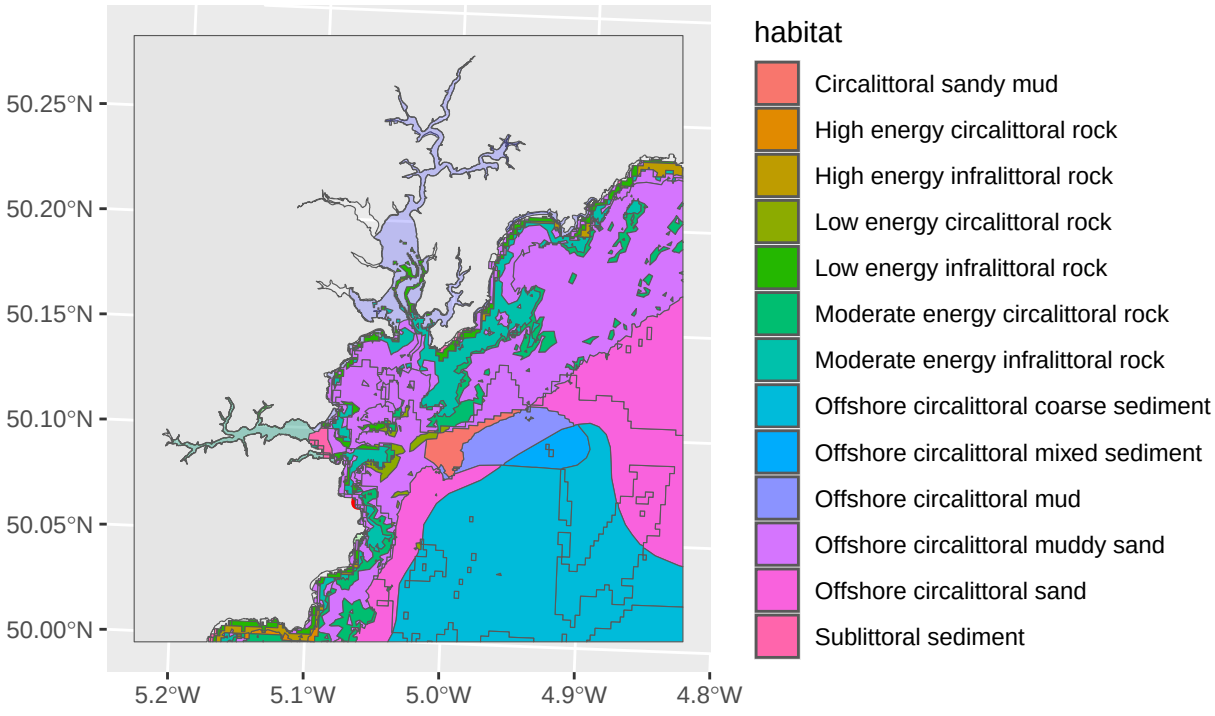
Let's start by reading in the UK Seamap benthic habitat map from our data folder. transform to the British National Grid and crop it to our study area.

```
habitats = st_read("data/uk_seamap.gpkg")
```

The UK Seamap dataset contains a lot of information and unintelligble field names. The habitat names we want are in the `JNCCName` column, so let's select just that field, rename it to `habitat`, and remove any features with missing (NA) values in the habitat column using the `tidyverse` function `drop_na`:

```
habitats = subset(habitats,select = JNCCName) %>%
           rename(habitat = JNCCName) %>%
           drop_na(habitat)

fan.plot + geom_sf(data = habitats, aes(fill = habitat))
```

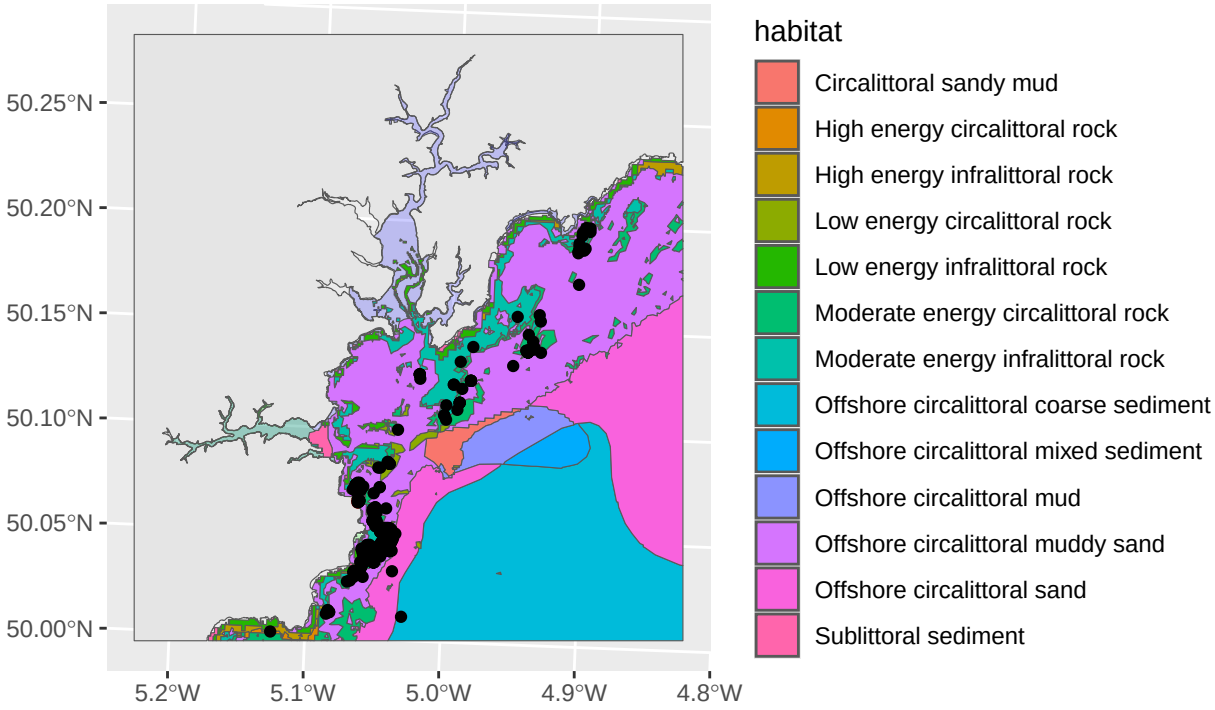


Looking at this map it appears there are multiple polygons for each habitat. If we inspect the `habitats` object there are 951 polygon features, but only 13 habitat types. We would like to union (or 'dissolve') the polygons for each habitat together to create a single `MULTIPOLYGON` for each. The function for unioning geometries is called `st_union`; but if just type `st_union(habitats)` we end up with a single large polygon containing the geometric union of all features. The easiest way to union geometries based on an attribute is to use the `group_by` and `summarise` functions from package `tidyverse`. First we group our data by `habitat` and then we summarise it, speciying `do_union = TRUE` (the default):

```
habitats = group_by(habitats,habitat) %>%
           summarise(do_union=TRUE)
```

That's worked nicely. We now have 13 multipolygons, one for each habitat. Let's plot the unioned habitats and add our pink sea fan sightings back on top.

```
fan.plot +
  geom_sf(data = habitats, aes(fill = habitat)) +
  geom_sf(data=fb.fans)
```



It appears from this plot that our pink sea fan observations are clustered around one particular habitat class (the greenish one), but let's test this properly using a "spatial join".

6. Performing spatial joins using `st_join()`

A spatial join involves joining the attributes from one spatial layer to another based on the geometric relationship between the two. In `sf` they are performed using a function called `st_join`. In this case, we want to join each point in our `fb.fans` dataset to the attributes of the `habitats` that it falls within, or intersects (note: as with spatial queries, `st_join` uses `st_intersects` as its default topological relation for joining so we don't need to specify it here):

```
fb.fans = st_join(fb.fans, habitats)

fb.fans
```

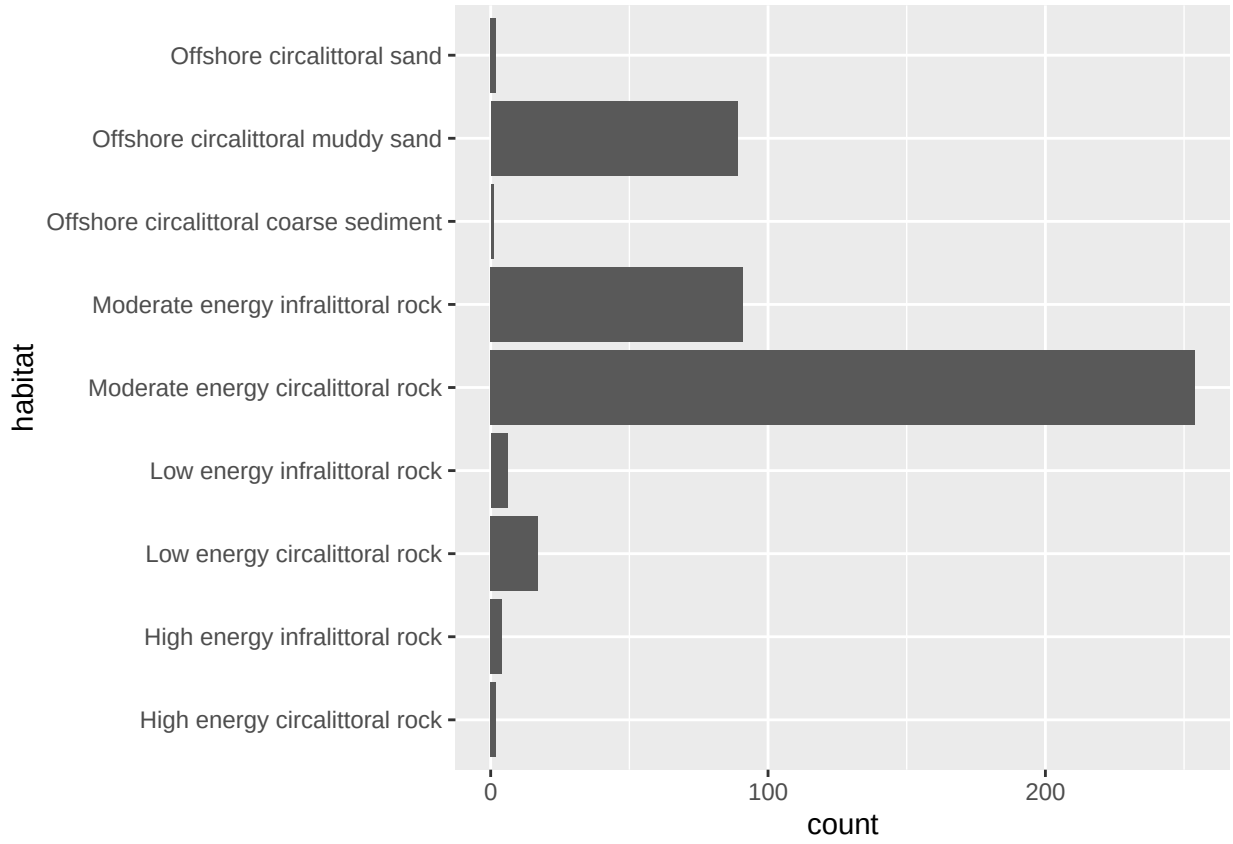
You can see that a new `habitat` column has now been added to `fb.fans`. We can count how many pink sea fan observations occurred in each habitat type using the `table` function to count appearances of each habitat in `fb.fans`

```
table(fb.fans$habitat)

# And work out the proportion
table(fb.fans$habitat)/nrow(fb.fans)
```

We could also create a barchart using `ggplot` to visualize the results. We use `geom_bar` to create a barchart and set the 'habitat' column as the x-axis variable to summarize. We can also flip the barchart on its side using `coord_flip` to make it easier to read:

```
ggplot() +
  geom_bar(data = fb.fans, aes(x = habitat)) +
  coord_flip()
```



This suggests there is a strong preference for "Moderate energy circalittoral rock", which contains about 54% of the sightings. But is this more than would expect by chance? If 54% of the seabed of Falmouth Bay consists of moderate energy circalittoral rock, this may be what you would expect from an entirely random distribution. We could check this by finding the area of each habitat type in Falmouth Bay using `st_area` and calculating the proportion:

```
st_area(habitats)/sum(st_area(habitats))
```

```
## Units: [1]
```

```
##  [1] 0.012655351 0.002172253 0.009193550 0.008544317 0.023028358 0.049288814
##  [7] 0.059237962 0.304920764 0.016180994 0.027458003 0.295904804 0.185991609
## [13] 0.005423220
```

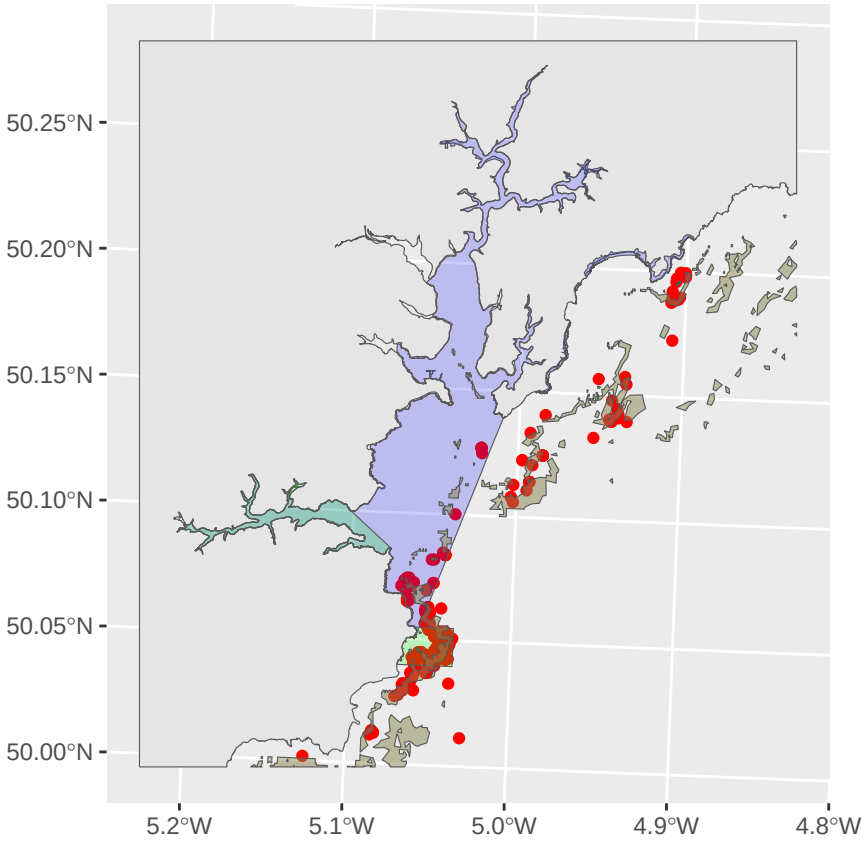
Circalittoral rock only comprises 5% of the total habitat area but contains > 50% of our observations, which suggests a preference. Note: this is a crude approach and in a real analysis we would want to test this more formally using something like a species distribution model. However, this result is fully conistent with what we understand about the ecology of the Pink Sea Fan, which is a filter feeder requiring hard substrate for attachment and a reasonably energetic environment with adequate water flow.

7. And finally: How effective are existing MPAs at protecting suitable habitat for pink sea fans?

So, what proportion of this key habitat is contained within existing MPAs? First we will subset out only our Moderate Energy Circalittoral Rock polygons and plot them:

```
rock = subset(habitats,habitat == "Moderate energy circalittoral rock")

fan.plot + geom_sf(data = rock,fill = 'khaki4', alpha=.5)
```



We can find the areas of rock habitat that fall inside MPAs using the `st_intersection` function we introduced earlier:

```
mcz.rock = st_intersection(rock,mcz)
sac.rock = st_intersection(rock,sac)
```

We can then find the area of suitable rocky habitat that intersects each class of MPA, and express this as a proportion of total habitat area in Falmouth Bay

```
st_area(mcz.rock)/st_area(rock)
st_area(sac.rock)/st_area(rock)
```

This tells us that around 12.5% of moderate energy circalittoral rock is protected in MPAs, including 8% in MCZs and 4.5% in the Falmouth & Helford SAC. It's more than the 10% Aichi target, but not great. Overall, our results are consistent with the conclusions of Pikesley et al. (2016) that Falmouth Bay is one of the areas where spatial protection of the Pink Sea Fan is lacking somewhat.

Taking it further: harder optional exercises if you want to challenge yourself!

1. One of the criticisms of our analysis is that sightings will be strongly biased towards areas that are visited often (e.g. popular dive sites). That is, the effort is very uneven. Many pink sea fan sightings will relate to the same locations and potentially even the same fans. To reduce this bias, Pikesley et al. (2016) used the number of 'Occupied Dive Sites' rather than individual sightings as their unit of measurement for assessing spatial overlap with MPAs. Information on the location of each sighting is provided in the `Locality` column of our `fans` dataset. Using the `st_union` function and the `st_centroid` function (which we haven't introduced yet), see if you can merge the sightings for each locality into a single point feature and rerun some of the analyses above. What difference does it make? Does this fully deal with the problems of spatial bias?

2. An opportunity has arisen to survey some more areas in Falmouth Bay using a drop down camera system to improve our knowledge of pink sea fan distributions. See if you can identify patches of suitable habitat with an area $>$ 10 hectare (100,000 m^2) that do not currently have any confirmed sightings. Then, using the `st_point_on_surface` function, find the coordinates of a point located within each patch that you can pass to the survey team as priorities for sampling. (Tips: This is hard, but can be done in a few lines of code. You will need to ensure that each habitat patch is a single `POLYGON` feature, not a `MULTIPOLYGON`, and use a spatial query to find polygons that don't intersect a point).

3. Using the editing tools in QGIS, modify the boundaries of the Manacles MCZ and the Falmouth & Helford SAC to increase spatial protection of pink sea fans and their key habitats. What is the greatest increase in coverage you can achieve for the smallest increase in area (bearing in mind the prerogative to keep MPA boundaries simple and easy to comply with for marine users).