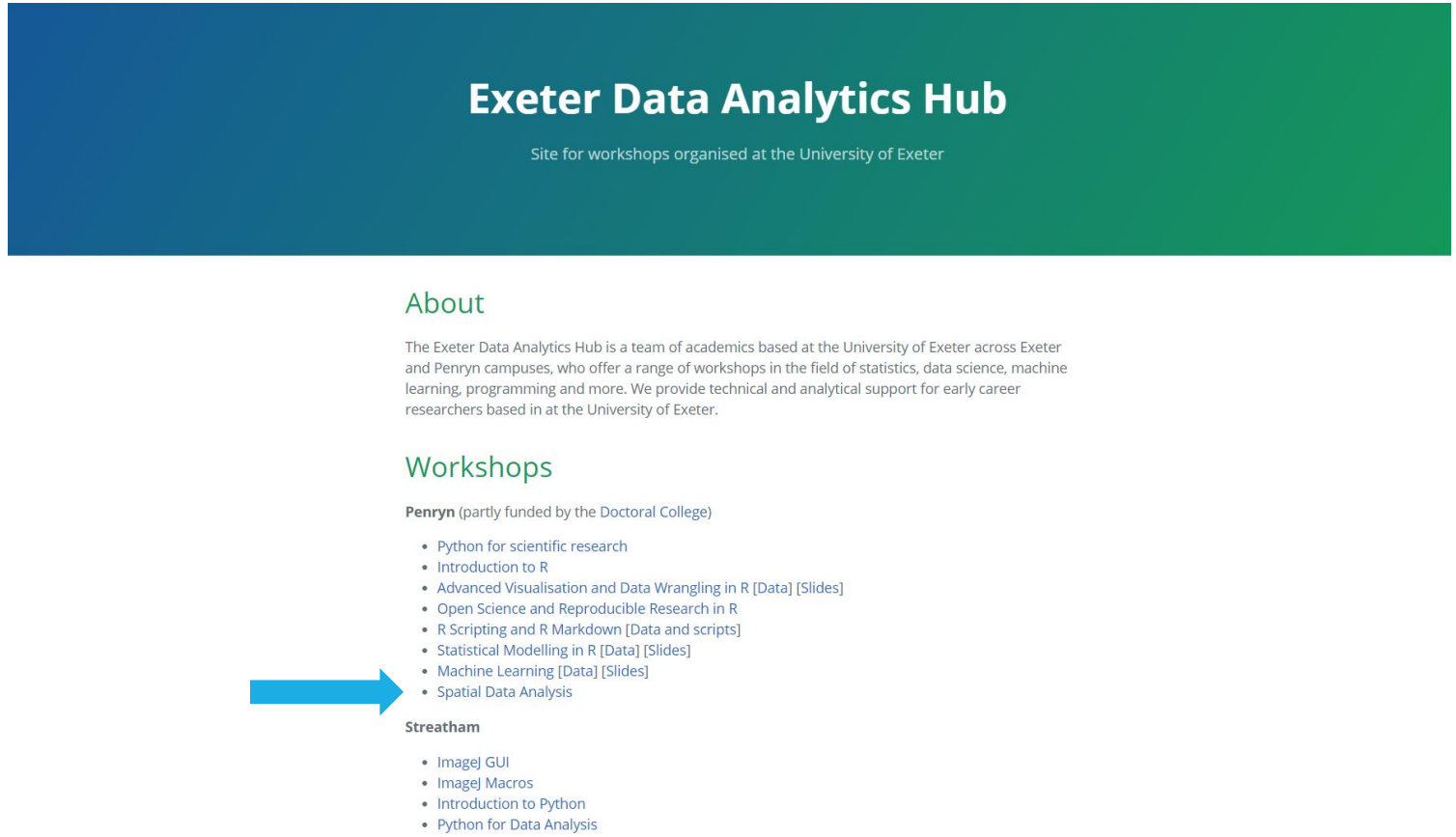
INTRODUCTION TO SPATIAL DATA ANALYSIS
WORKING WITH RASTER DATA IN R

Chris Yeomans (c.m.yeomans@exeter.ac.uk)
David March (d.march@exeter.ac.uk)
James Duffy (j.duffy@exeter.ac.uk)

Penryn Campus, 20th & 27th November 2019

COURSE MATERIALS

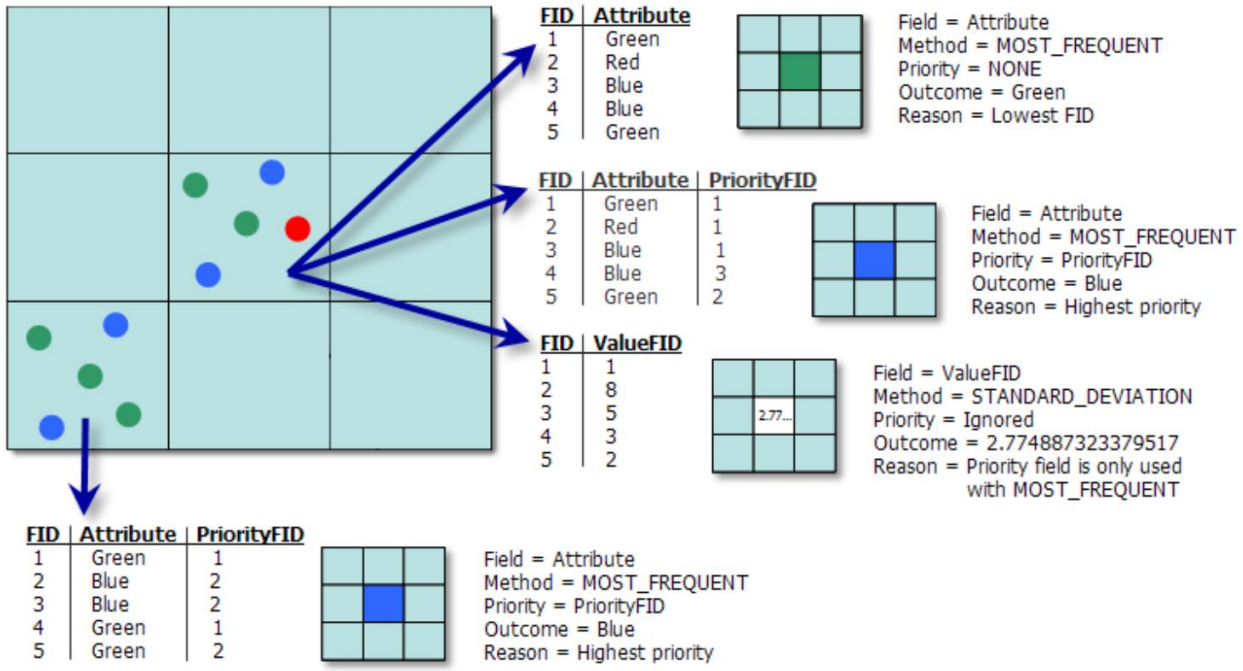
https://exeter-data-analytics.github.io/



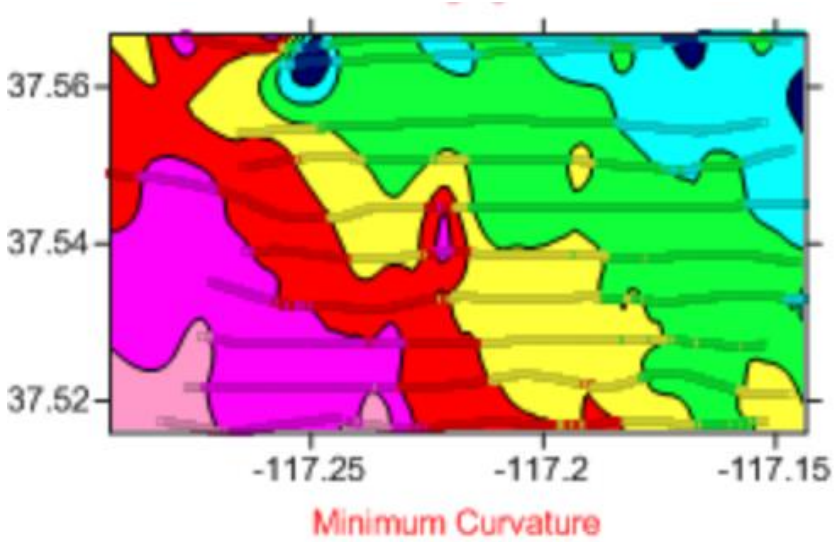
REAL WORLD TO RASTER MODEL

RASTER DATA COLLECTION

Cell assignment



Gridding/Interpolation/Spline



Picture Elements or pixels

Sources: ArcGIS Pro; Golden Software; Google

LEARNING BY HANDS-ON PRACTICAL SESSIONS

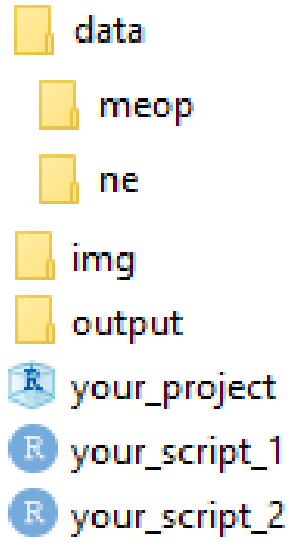
Contents

- Import spatial data in *R* (raster)
- Generating synthetic data
- Real examples
- Raster operations
- Raster manipulation
- Visualize maps
- Save data

- Time to explore other methods or your own data

SET UP YOUR PROJECT

1. Create a new project from Rstudio
2. Download the Data
3. Unzip and store its contents in a folder named data/ within your new project.
4. Create folders output/ and img/
5. Install all required packages



RASTER DATA FORMATS

Getting raster data in is easy and can be done using the following commands

```
r1 <- raster()
s1 <- stack()
```

The `raster()` tool is used for single layer files and the `stack()` tool for raster datasets with multiple layers, or for combining single layer rasters into a multilayer raster.

IMPORT RASTER DATA

4.2.1 Importing a raster layer

A single raster layer is easily imported. Here we load an example from the system repository.

```
r1 <- raster(system.file("external/test.grd", package="raster"))
```

IMPORT RASTER DATA

4.2.2 Importing a raster stack

Rasters can be stacked and this is particularly useful for RGB layers in a raster. For example, we can illustrate this using the R logo:

```
sRLogo <- stack(system.file("external/rlogo.grd", package="raster"))
nlayers(sRLogo)
```

PLOTTING RASTER DATA QUICKLY

There are three layers in the `sRLogo` representing the red, green and blue channels (RGB). These can be plotted individually.

```
image(sRLogo, y = 1)
```

```
image(sRLogo$green)
```

```
image(sRLogo[[3]])
```

PLOTTING RASTER DATA QUICKLY

These data may be more useful if plotted as an RGB plot. R isn't great at this but for a quick visualisation we can use `plotRGB`:

```
plotRGB(sRLogo, stretch = "lin")
```

There are few tricks to `plotRGB` worth exploring such as changing the stretch:

```
plotRGB(sRLogo, stretch = "hist")
```

And changing the order of the bands - useful when RGB are not in the correct order:

PLOTTING RASTER DATA QUICKLY

4.2.2 Importing a raster stack

Rasters can be stacked and this is particularly useful for RGB layers in a raster. For example, we can illustrate this using the R logo:

```
sRLogo <- stack(system.file("external/rlogo.grd", package="raster"))
nlayers(sRLogo)
```

CREATING SYNTHETIC DATA

Sometimes it can be useful to generate a random synthetic dataset. This could be for adding random noise to a dataset or for simple testing.

```
rSynth <- raster()
set.seed(0)
values(rSynth) <- runif(ncell(rSynth), 0, 100)
```

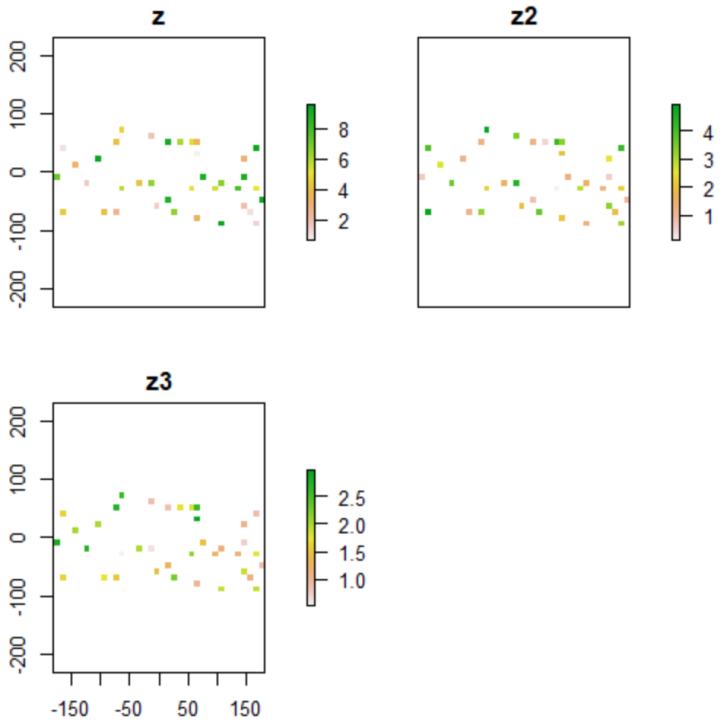
The success of this can be checked in a number of ways:

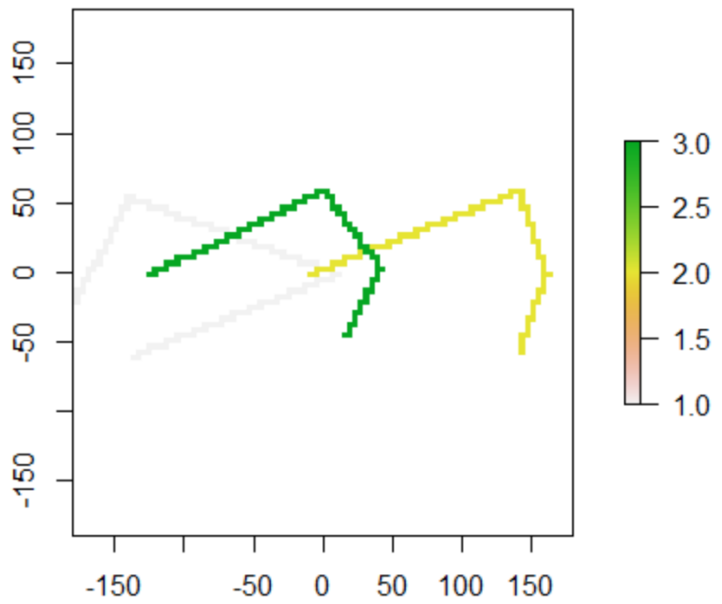
```
hasValues(rSynth)
length(values(rSynth))
ncell(rSynth)
```

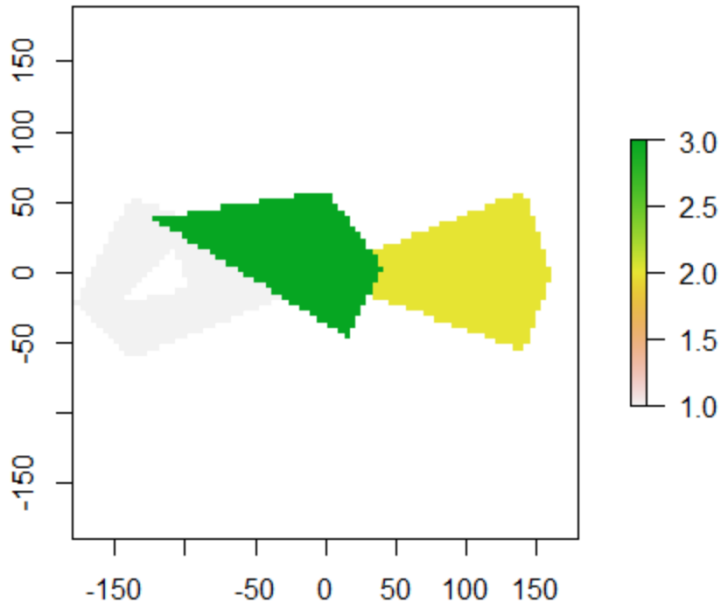
Similarly, a synthetic raster stack can be generated by simply manipulating the:

```
sSynth <- stack(rSynth, rSynth*2, sqrt(rSynth))
plot(sSynth[[2]])
```

RASTERISING VECTOR DATA







REAL EXAMPLES – ELEVATION DATA

```
cwl <- readOGR("data/ukdata/england_ct_2001.shp")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Git\spatial_data_analysis\data\ukdata\england_ct_2001.shp", layer: "england_ct_2001"
## with 18 features
## It has 2 fields
```

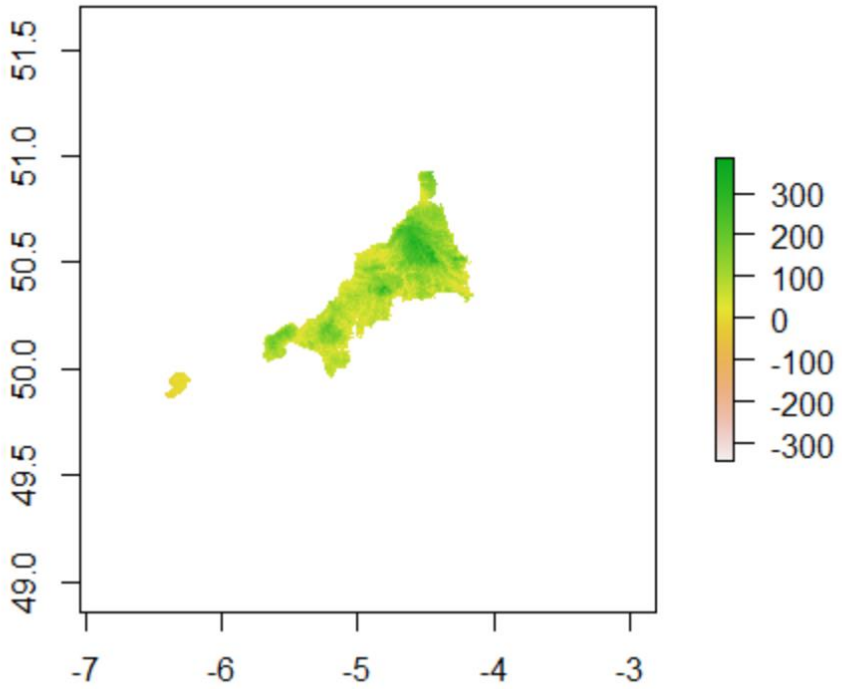
```
wgs84 = '+proj=longlat +datum=WGS84'
cwl_WGS84 <- spTransform(cwl, CRS(wgs84))
```

The elevation data is accessed using the `elevatr` package. more details can be found here

```
elev_cwl_WGS84 <- get_elev_raster(cwl_WGS84, z=9)
```

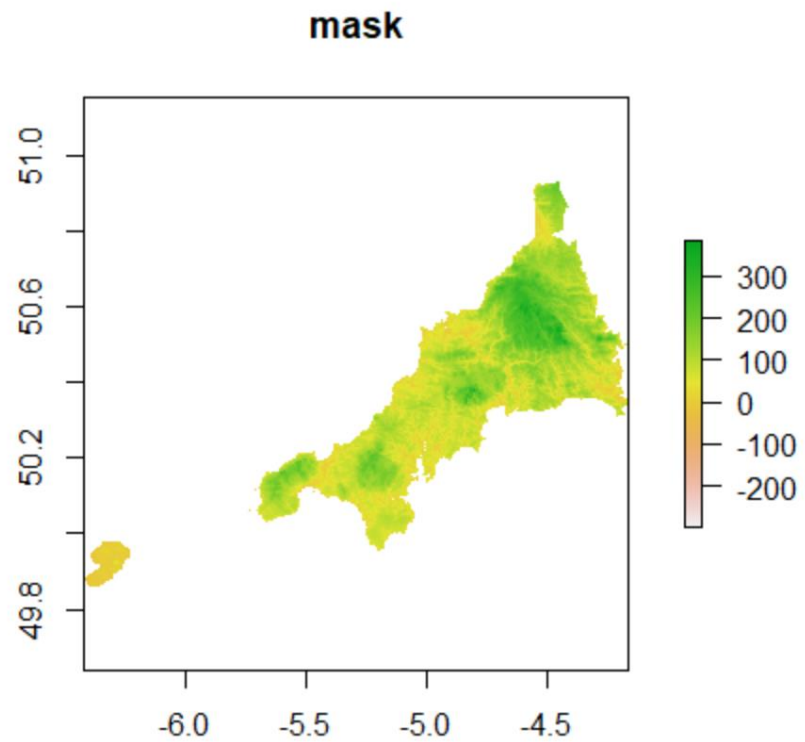
RASTER OPERATIONS – MASK

```
mask_elev <- mask(elev_cwl_WGS84, cwl_WGS84)
plot(mask_elev)
```



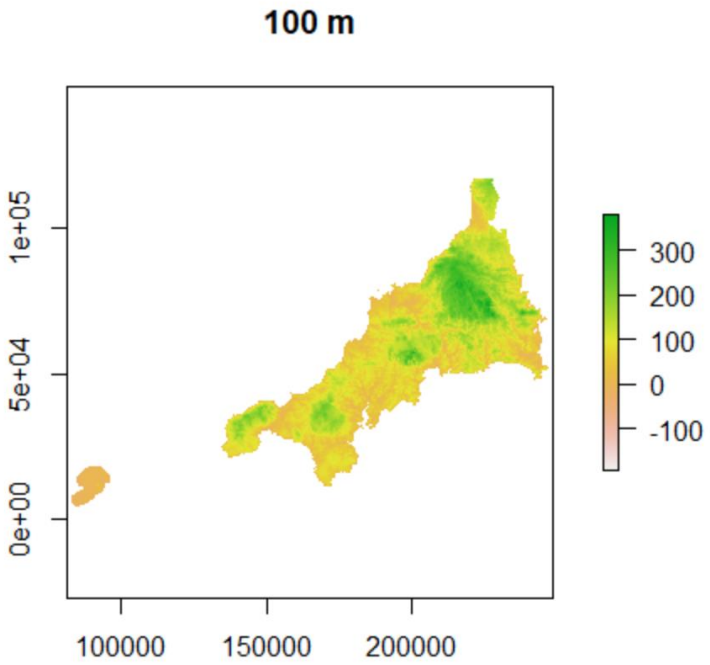
RASTER OPERATIONS – CROP

```r
e <- extent(cwl_WGS84)
crop_mask_elev <- crop(mask_elev, e)
plot(crop_mask_elev, main="mask")
```



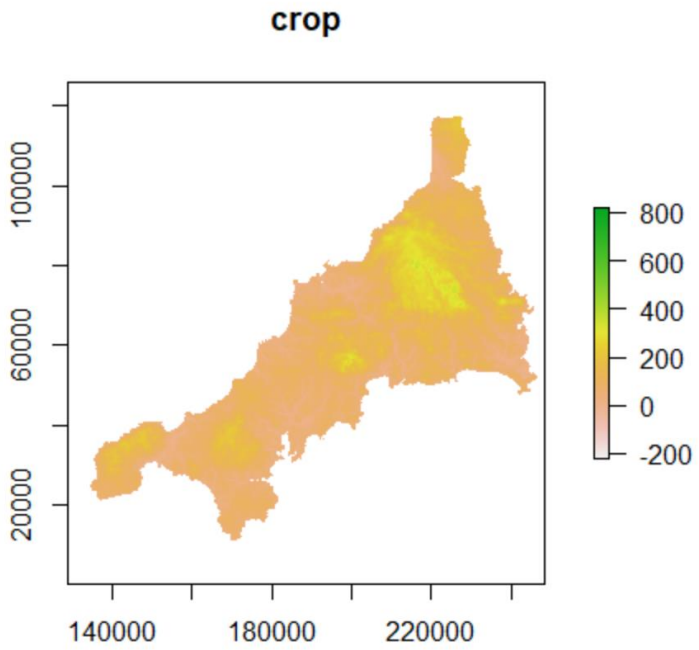
RASTER OPERATIONS – RESAMPLE

```
tmpR <- raster(nrow=1271,ncol=1669)
crs(tmpR) <- crs(cwl)
extent(tmpR) <- c(81600,248500,-2600,124500)
elev_100m <- resample(elev_cwl_BNG, tmpR)
plot(elev_100m,main="100 m")
```



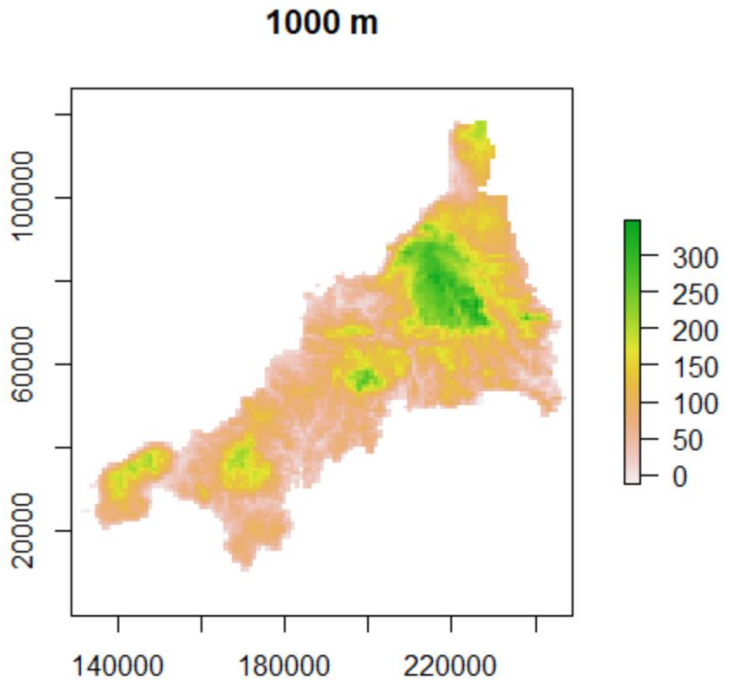
RASTER OPERATIONS – EXTRACT

```r
# For example you could try this... but give the locator() tool a go
e_crop <- c(129000, 248500, 1600, 124500) # xmin, xmax, ymin, ymax
# Now crop
crop_elev_100m <- crop(elev_100m, e_crop)
plot(crop_elev_100m, main="crop")
```



RASTER OPERATIONS – AGGREGATE

```
### Aggregate to a lower res raster
elev_1000m <- aggregate(crop_elev_100m,10,fun=mean,expand=TRUE, na.rm=TRUE)
plot(elev_1000m, main = "1000 m")
```



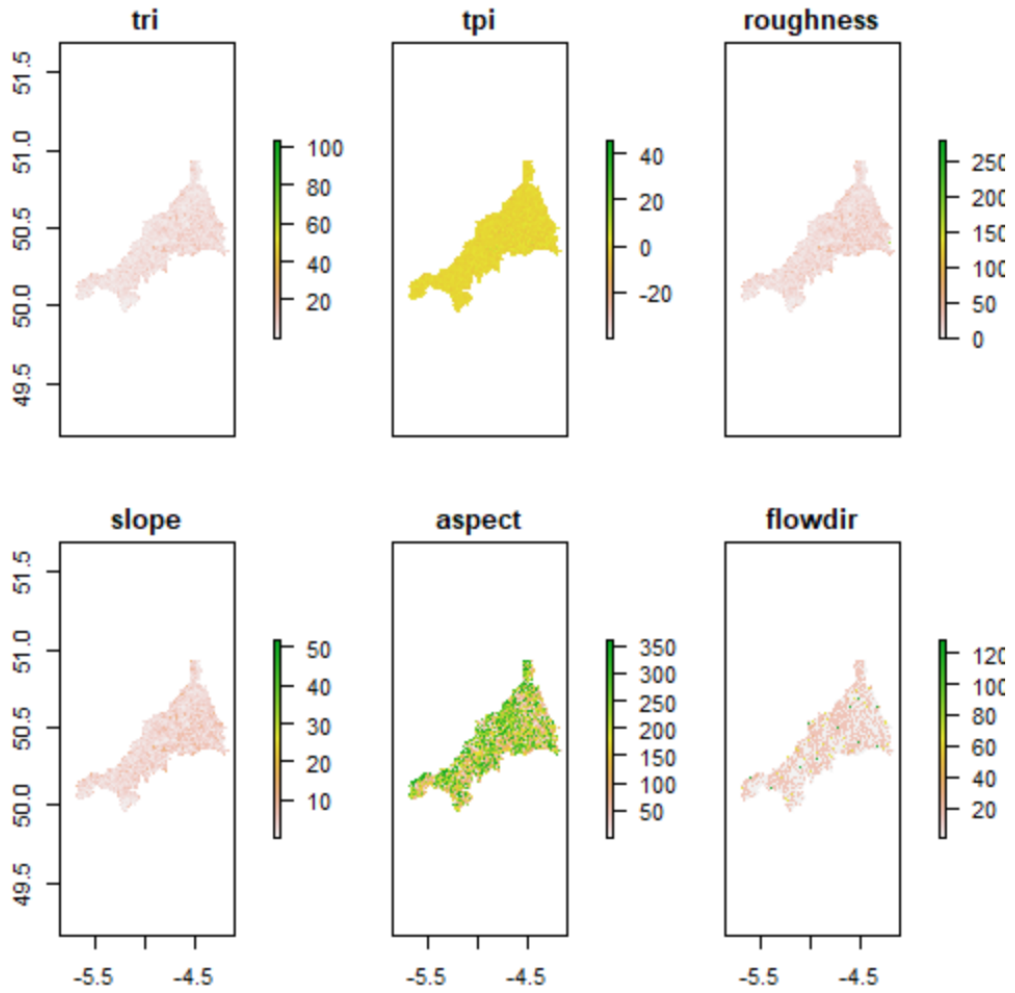
RASTER MANIPULATION

The `cellStats` function is useful for finding broad statistics about a layer e.g. sum, mean, min, max, sd, ‘skew’ and ‘rms’ of which the latter two must be supplied as a character value (with quotes)

```
cellStats(elev_100m_WGS84, stat='mean')
```

RASTER MANIPULATION

```
models <- terrain(elev_100m_WGS84, opt=
class(models)
```



RASTER MANIPULATION – HILLSHADE

We an also make a hillshade raster and overlay the elevation data. We introduce a way of subsetting the raster stack/brick here using `subset` but you could also use `slope <- models$slope` or other means we introduced at the beginning of this section.

```
slope <- subset(models, "slope")  # select slope
aspect <- subset(models, "aspect")  # select slope
hill <- hillShade(slope, aspect, 45, 270)
plot(hill, col = grey(0:100/100), legend = FALSE)
plot(elev_100m_WGS84, col = rainbow(25, alpha=0.35), add=TRUE)
```

RASTER VISUALISATION

Maps can be quickly created using `leaflet` and basemaps from OpenStreetMap or tiles from MapBox imagery can be imported to enhance the map.

```
pal <- colorNumeric(c("#FFFFCC", "#41B6C4", "#0C2C84"), domain=c(0,500, na.rm=T),
                    na.color = "transparent", reverse = T)
leaflet() %>%
  addTiles() %>%  # OpenStreetMap base map
  addRasterImage(elev_100m_WGS84, colors = pal, opacity = 0.8) %>%
  addLegend(pal = pal, values = c(0, 500),
            title = "Cornwall Elevation")
```

SAVING RASTER FILES

Now we can write our rasters as layers, stacks or write layers from stacks…

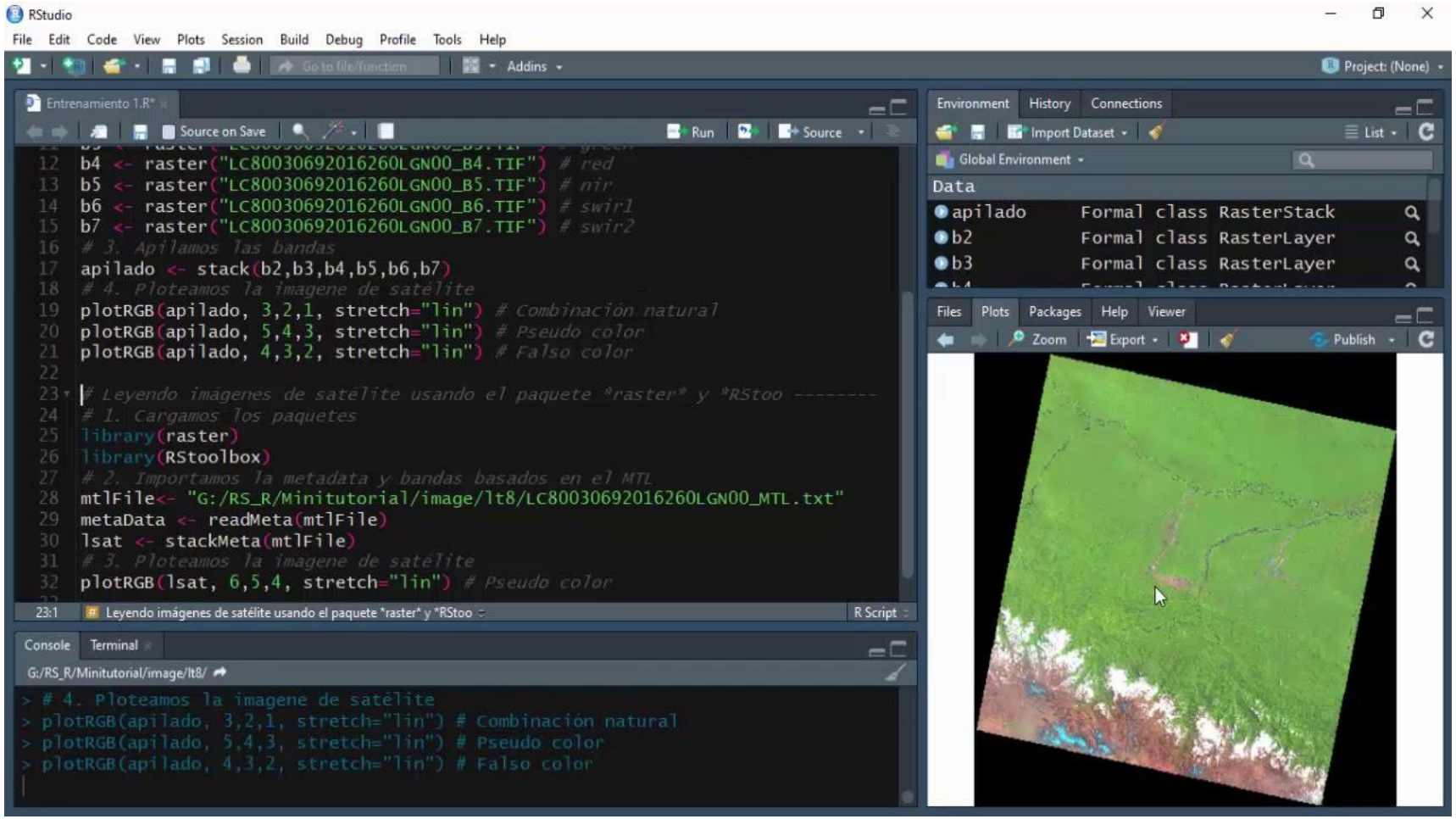
```
# A raster layer
writeRaster(slope, filename="output/slope.tif", format="GTiff", overwrite=TRUE)
# A layer from a raster stack/brick
writeRaster(models[[1]], filename="output/tri", format="GTiff", overwrite=TRUE)
# A whole stack/brick
writeRaster(models, filename="output/models", format="GTiff", overwrite=TRUE)
# A whole raster stack/brick by layer
writeRaster(models, filename=paste("output/", names(models), sep = ""), format="GTiff", bylayer=T, overwrite=TR
```

Or you can write it to KML with a set colour palette

```
myPal <- colorRampPalette(brewer.pal(9, 'Blues'), alpha=TRUE)  # palette
KML(elev_100m_WGS84, "output/elev_100m_WGS84.kml", col = myPal(100), overwrite = TRUE)
```

ADDITIONAL APPLICATIONS

REMOTE SENSING

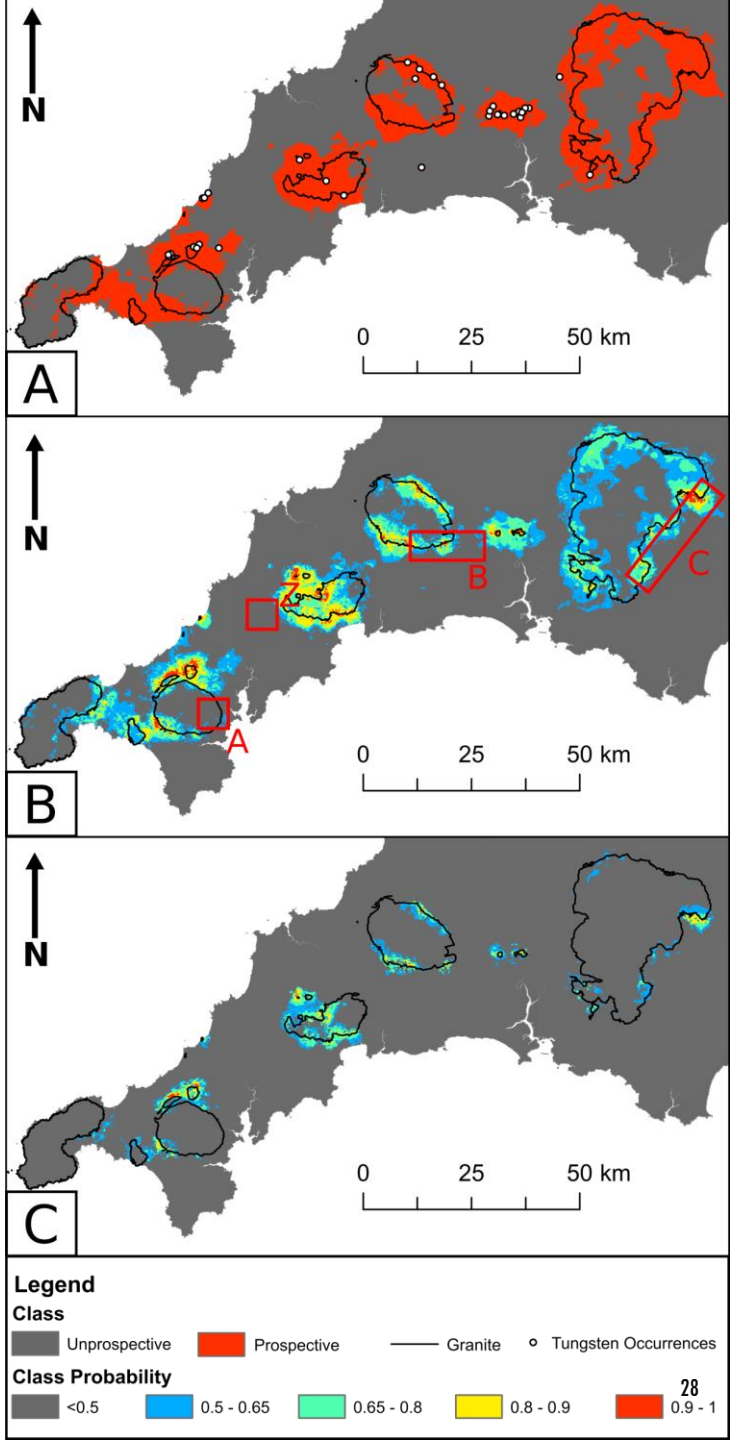


[Landsat in R]

[Sentinel-2 in R]

MLA APPLICATIONS

```
#### Random Forest Prospectivity ####
####Manipulate to training data to dataframe
df.Wtr <- as.data.frame(Wtr@data)
# Change response col name to match training function which is "class"
colnames(df.Wtr) <- "class"
# Check
View(df.Wtr)
# Stick it back into the spatial points file
Wtr@data <- df.Wtr
#### Random Forest Model Training
# Model for classification
mod1 <- rasterRFmodel(Wr.mg, Wtr,cp="raw",imp=T,ntree=20000)
# Model for Probability
mod2 <- rasterRFmodel(Wr.mg, Wtr,cp="prob",imp=T,ntree=20000)
#### Random Forest Prediction
# Predict model for classification
rfp1.mod1 <- rasterRFpredict(r=Wr.mg, mod=mod1,cp="raw",imp=T,ntree=20000)
# For some reason the script won't write this to raster layer, only to a brick
# so convert to raster layer
rfp1.mod1 <- rfp1.mod1[[1]]
# Predict model for probability
rfp1.mod2 <- rasterRFpredict(r=Wr.mg, mod=mod2,cp="prob",imp=T,ntree=20000)
#### Model Confidence for prosecptive class (Class 2)
uncert.mod2 <- rasterUncert(rfp1.mod2)
rfconf <- rfp1.mod2[[2]] - uncert.mod2/(nlayers(rfp1.mod2))
rfconfnorm <- (rfconf-cellStats(rfconf, "min"))/
                (cellStats(rfconf, "max")-cellStats(rfconf, "min"))
plot(rfconfnorm, main = "RF confidence")
```



ACKNOWLEDGEMENTS

- JJ Valletta & TJ McKinley - for their time and efforts in setting up the data analytics program
- CY acknowledges funding from NERC as part of the GWatt project
- DM and JD are thanked for creating the Spatial Data Analytics repo