Preparatory and exploratory data analysis for digital soil mapping

Soil Security Laboratory

2017

At the start of this book, some of the history and theoretical underpinnings of DSM was discussed. Now with a solid foundation in `R`, it is time to put this all into practice i.e. do DSM with `R`.

In this chapter some common methods for soil data preparation and exploration are covered. Soil point databases are inherently heterogeneous because soils are measured non uniformly from site to site. However one more-or-less commonality is that soil point observations will generally have some sort of label, together with some spatial coordinate information that indicates where the sample was collected from. Then things begin to vary from site to site. Probably the biggest difference is that all soils are not measured universally at the same depths. Some soils are sampled per horizon or at regular depths. Some soil studies examine only the topsoil, while others sample to the bedrock depth. Then different soil attributes are measured at some locations and depths, but not at others. Overall, it becomes quickly apparent when one begins working with soil data that a number of preprocessing steps are needed to fulfill the requirements of a particular analysis.

In order to prepare a collection of data for use in a DSM project as described in Minasny and McBratney (2010) one needs to examine what data are available, what is the soil attribute or class to be modeled? What is the support of the data? This includes whether observations represent soil point observations or some integral over a defined area (for now we just consider observations to be point observations). However, we may also assume the support to be also a function of depth in that we may be interested in only mapping soil for the top 10cm, or to 1m, or any depth in between or to the depth to bedrock. The depth interval could be a single value (such as one value for the 0-1m depth interval as an example), or we may wish to map simultaneously the depth variation of the target soil attribute with the lateral or spatial variation. These questions add complexity to the soil mapping project, but are an important consideration when planning a project and assessing what the objectives are.

More recent digital soil mapping research has examined the the combination of soil depths functions with spatial mapping in order to create soil maps with a near 3-D support. In the following section some approaches for doing this are discussed with emphasis and instruction on a particular method, namely the use of a mass-preserving soil depth function. This will be followed by a section that will examine the important DSM step of linking observed soil information with available environmental covariates and the subsequent preparation of covariates for spatial modelling.

1 Soil depth functions

The traditional method of sampling soil involves dividing a soil profile into horizons. The number of horizons and the position of each are generally based on attributes easily observed in the field, such as morphological soil properties (Bishop et al., 1999). From each horizon, a bulk sample is taken and it is assumed to represent the average value for a soil attribute over the depth interval from which it is sampled. There are some issues with this approach, particularly from a pedological perspective and secondly from the difficulty in using this legacy data within a Digital Soil Mapping (DSM) framework where we wish to know the continuous variability of a soil both in the lateral and vertical dimensions. From the pedological perspective soil generally varies continuously with depth; however, representing the soil attribute value as the average over the depth interval of horizons leads to discontinuous or stepped profile representations. Difficulties can arise in situations where one wants to know the value of an attribute at a specified depth. The second issue is regarding DSM and is where we use a database of soil profiles to generate a model of soil variability in the area in which they exist. Because observations at each horizon for each profile will rarely be the same between any two profiles, it then becomes difficult to build a model where predictions are made at a set depth or at standardized depth intervals.

Legacy soil data is too valuable to do away with and thus needs to be molded to suit the purposes of the map producer, such that one needs to be able to derive a continuous function using the available horizon data as some input. This can be done with many methods including polynomials and exponential decay type depth functions. A more general continuous depth function is the equal-area quadratic spline function. The usage and mathematical expression of this function have been detailed in Ponce-Hernandez et al. (1986); Bishop et al. (1999), and Malone et al. (2009). A useful feature of the spline function is that it is mass preserving, or in other words the original data is preserved and can be retrieved again via integration of the continuous spline. Compared to exponential decay functions where the goal is in defining the actual parameters of the decay function, the spline parameters are the values of the soil attribute at the standard depths that are specified by the user. This is a useful feature, because firstly, one can harmonize a whole collection of soil profile data and then explicitly model the soil for a specified depth. For example the GlobalSoilMap.net project (Arrouays et al., 2014) has a specification that digital soil maps be created for each target soil variable for the 0-5cm, 5-15cm, 15-30cm, 30-60cm, 60-100cm, and 100-200cm depth intervals. In this case, the mass-preserving splines can be fitted to the observed data, then values can be extracted from them at the required depths, and are then ready for exploratory analysis and spatial modelling.

In the following, we will use legacy soil data and the spline function to prepare data to be used in a DSM framework. This will specifically entail fitting splines to all the available soil profiles and then through a process of harmonization, integrate the splines to generate harmonized depths of each observation.

1.1 Fit mass preserving splines with R

We will demonstrate the mass-preserving spline fitting using a single soil profile example for which there are measurements of soil carbon density to a given maximum depth. We can fit a spline to the maximum soil depth, or alternatively any depth that does not exceed the maximum soil depth. The function used for fitting splines is called `ea_spline` and is from the `ithir` package. Look at the help file for further information on this function. For example, there is further information about how the `ea_spline` function can also accept data of class `SoilProfileCollection` from the `aqp` package in addition to data of the more generic `data.frame` class. In the example below the input data is of class `data.frame`. The data we need (`oneProfile`) is in the `ithir` package. Lets also load a few of the other important `R` packages that will be encountered in this section too

```
# Load in libraries
library(ithir)
library(raster)
library(sp)
library(gstat)
library(nortest)
library(fBasics)

# ithir dataset
data(oneProfile)
str(oneProfile)
```

```
## 'data.frame': 8 obs. of  4 variables:
##  $ Soil.ID       : num  1 1 1 1 1 1 1 1
##  $ Upper.Boundary: num  0 10 30 50 70 120 250 350
##  $ Lower.Boundary: num  10 20 40 60 80 130 260 360
##  $ C.kg.m3.      : num  20.7 11.7 8.2 6.3 2.4 2 0.7 1.2
```

As you can see above, the data table shows the soil depth information and carbon density values for a single soil profile. Note the discontinuity of the observed depth intervals which can also be observed in Figure 1.

The `ea_spline` function will predict a continuous function from the top of the soil profile to the maximum soil depth, such that it will interpolate values both within the observed depths and between the depths where there is no observation.

To parametize the `ea_spline` function, we could accept the defaults, however it might be likely to change the `lam` and `d` parameters to suit the objective of the analysis being undertaken. `lam` is the lambda parameter which controls the smoothness or fidelity of the spline. Increasing this value will make the

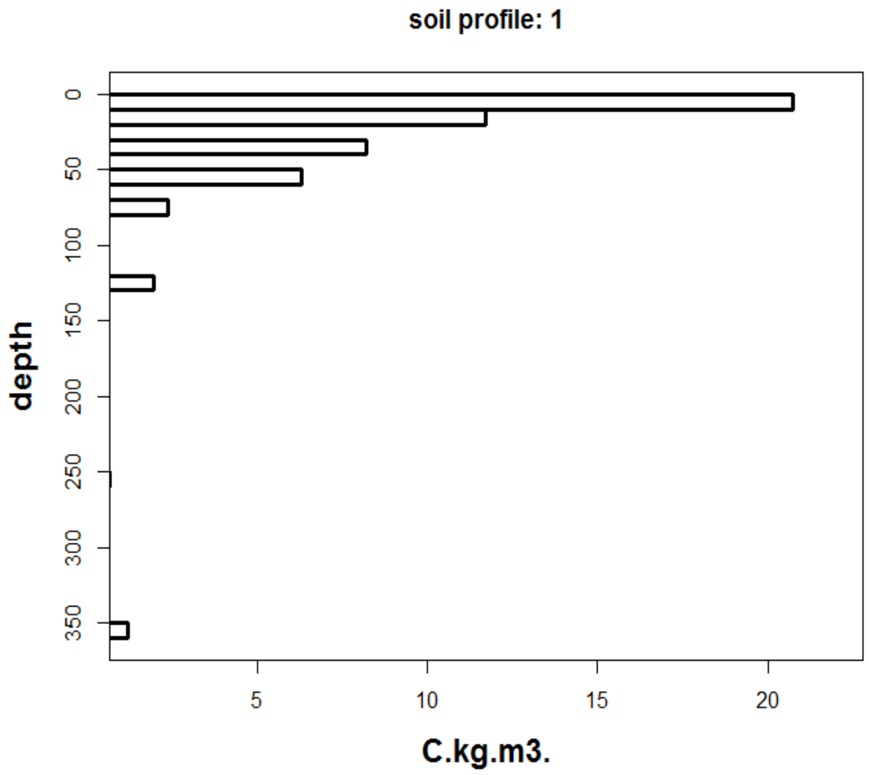

Figure 1: Soil profile plot of the `oneProfile` data. Note this figure was produced using the `plot_soilProfile` function from `ithir`.

spline more rigid. Decreasing it towards zero will make the spline more flexible such that it will follow near directly the observed data. A sensitivity analysis is generally recommended in order to optimize this parameter. From experience a `lam` value of 0.1 works well generally for most soil properties, and is the default value for the function. The `d` parameter represents the depth intervals at which we want to get soil values for. This is a harmonization process where regardless of which depths soil was observed at, we can derive the soil values for regularized and standard depths. In practice, the continuous spline function is first fitted to the data, then we get the integrals of this function to determine the values of the soil at the standard depths. `d` is a matrix, but on the basis of the default values, what it is indicating is that we want the values of soil at the following depth intervals: 0-5cm, 5-15cm, 15-30cm, 30-60cm, 60-100cm, and 100-200cm. These depths are specified depths determined for the GlobalSoilMap.net project (Arrouays et al., 2014). Naturally, one can alter these values to suit there own particular requirements. To fit a spline to the carbon density values of the `oneProfile` data, the following script could be used:

```
eaFit <- ea_spline(oneProfile, var.name = "C.kg.m3.",
d = t(c(0, 5, 15, 30, 60, 100, 200)), lam = 0.1, vlow = 0,
show.progress = FALSE)
```

```
## Fitting mass preserving splines per profile...
```

```
str(eaFit)
```

```
## List of 4
##  $ harmonised:'data.frame': 1 obs. of  8 variables:
##   ..$ id         : num 1
##   ..$ 0-5 cm     : num 21
##   ..$ 5-15 cm    : num 15.8
##   ..$ 15-30 cm   : num 9.89
##   ..$ 30-60 cm   : num 7.18
##   ..$ 60-100 cm  : num 2.76
##   ..$ 100-200 cm: num 1.73
##   ..$ soil depth: num 360
##  $ obs.preds :'data.frame': 8 obs. of  6 variables:
##   ..$ Soil.ID       : num [1:8] 1 1 1 1 1 1 1 1
##   ..$ Upper.Boundary: num [1:8] 0 10 30 50 70 120 250 350
##   ..$ Lower.Boundary: num [1:8] 10 20 40 60 80 130 260 360
##   ..$ C.kg.m3.      : num [1:8] 20.7 11.7 8.2 6.3 2.4 2 0.7 1.2
##   ..$ predicted     : num [1:8] 19.84 12.45 8.24 6.2 2.56 ...
##   ..$ FID           : num [1:8] 1 1 1 1 1 1 1 1
##  $ var.1cm   : num [1:200, 1] 21.6 21.4 21.1 20.8 20.3 ...
##  $ tmse      : num [1, 1] 0.263
```

The output of the function is a `list`, where the first element is a `dataframe` (`harmonized`) which are the predicted spline estimates at the specified depth intervals. The second element (`obs.preds`) is another `dataframe` but contains the observed soil data together with spline predictions for the actual depths of observation for each soil profile. The third element (`var.1cm`) is a matrix which stores the spline predictions of the depth function at (in this case) 1cm resolution. Each column represent a given soil profile and each row represents an incremental 1cm depth increment to the maximum depth we wish to extract values for, or to the maximum observed soil depth (which ever is smallest). The last element (`tmse`) is another matrix but stores a single mean square error estimate for each given soil profile. This value is an estimate of the magnitude of difference between observed values and associated predicted values with each profile. It is often more amenable to visualize the performance of the spline fitting. Subsequently, plotting the outputs of `ea_spline` is made possible by the associated `plot_ea_spline` function (see help file for use of this function):

```
par(mfrow = c(3, 1))
for (i in 1:3) {
    plot_ea_spline(splineOuts = eaFit, d = t(c(0, 5, 15, 30, 60, 100, 200)),
        maxd = 200, type = i, plot.which = 1, label = "carbon density")
}
```

The `plot_ea_spline` function is a basic function without too much control over the plotting parameters, except there are three possible themes of plot output that one can select. This is controlled by the `type` parameter. `Type = 1` is to return the observed soil data plus the continuous spline (default). `Type = 2` is to return the observed data plus the averages of the spline at the specified depth intervals. `Type = 3` is to return the observed data, spline averages and continuous spline. The script above results in producing all three possible plots, and is shown on Figure 2.

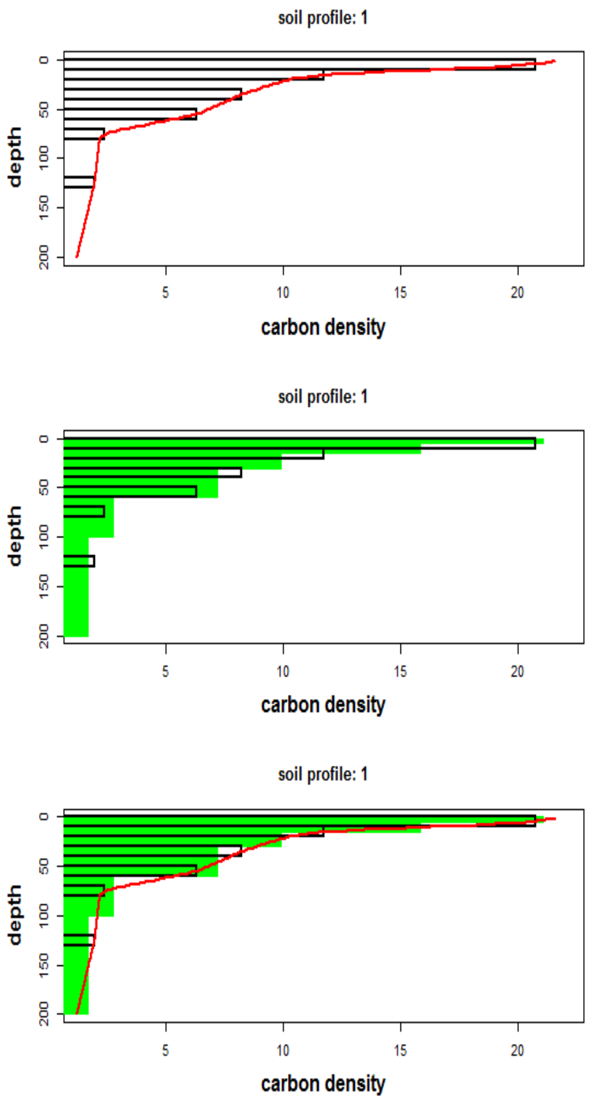

Figure 2: Soil profile plot of the three type variants from `plot_ea_spline`. Plot 1 is type 1, plot 2 is type 2 and plot 3 is type 3.

2 Intersecting soil point observations with environmental covariates

In order to carry out digital soil mapping in terms of evaluating the significance of environmental variables in explaining the spatial variation of the target soil variable under investigation, we need to link both sets of data together and extract the values of the covariates at the locations of the soil point data. The first task is to bring in to our working environment some soil point data. We will be using a data set of soil samples collected from the Hunter Valley, NSW Australia. The area in question is an approximately $22km^2$ sub-area of the Hunter Wine Country Private Irrigation District (HWCPID), situated in the Lower Hunter Valley, NSW (32.83S 151.35E). The HWCPID is approximately 140 km north of Sydney, NSW, Australia. The target variable is soil pH. The data was preprocessed such that the predicted values are outputs of the mass-preserving depth function. In the following examples we will just focus on the values of soil pH at the 60-100cm depth interval. The data is loaded in from the `ithir` package with the following script:

```
data(HV_subsoilpH)
str(HV_subsoilpH)
```

```
## 'data.frame': 506 obs. of  14 variables:
##  $ X                         : num  340386 340345 340559 340483 340734 ...
##  $ Y                         : num  6368690 6368491 6369168 6368740 6368964 ...
##  $ pH60_100cm                : num  4.47 5.42 6.26 8.03 8.86 ...
##  $ Terrain_Ruggedness_Index: num  1.34 1.42 1.64 1.04 1.27 ...
##  $ AACN                      : num  1.619 0.281 2.301 1.74 3.114 ...
##  $ Landsat_Band1             : int  57 47 59 52 62 53 47 52 53 63 ...
##  $ Elevation                 : num  103.1 103.7 99.9 101.9 99.8 ...
##  $ Hillshading               : num  1.849 1.428 0.934 1.517 1.652 ...
##  $ Light_insolation          : num  1689 1701 1722 1688 1735 ...
##  $ Mid_Slope_Positon         : num  0.876 0.914 0.844 0.848 0.833 ...
##  $ MRVBF                     : num  3.85 3.31 3.66 3.92 3.89 ...
##  $ NDVI                      : num  -0.143 -0.386 -0.197 -0.14 -0.15 ...
##  $ TWI                       : num  17.5 18.2 18.8 18 17.8 ...
##  $ Slope                     : num  1.79 1.42 1.01 1.49 1.83 ...
```

As you will note, there are 506 observations of soil pH. These data are assocaited with a spatial coordinate and also have associated environmental covariate data that have been intersected with the point data. The environmental covariates have been sourced from a digital elevation model and Landsat 7 satelite data. For the demonstration purposes of the exercise, we will firstly remove this already intersected data and start fresh - essentially this is an opportunity to recall earlier work on `data frame` manipulation and indexing.

```
# round pH data to 2 decimal places
HV_subsoilpH$pH60_100cm <- round(HV_subsoilpH$pH60_100cm, 2)
```

```
# remove already intersected data
HV_subsoilpH <- HV_subsoilpH[, 1:3]

# add an id column
HV_subsoilpH$id <- seq(1, nrow(HV_subsoilpH), by = 1)

# re-arrange order of columns
HV_subsoilpH <- HV_subsoilpH[, c(4, 1, 2, 3)]

# Change names of coordinate columns
names(HV_subsoilpH)[2:3] <- c("x", "y")
```

Also in the ithir package are a collection of rasters that correspond to environmental covariates that cover the extent of the just loaded soil point data. These can be loaded using the script:

```
data(hunterCovariates_sub)
hunterCovariates_sub
```

```
## class       : RasterStack
## dimensions  : 249, 210, 52290, 11  (nrow, ncol, ncell, nlayers)
## resolution  : 25, 25  (x, y)
## extent      : 338422.3, 343672.3, 6364203, 6370428  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=56 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs
## names       : Terrain_Ruggedness_Index,        AACN, Landsat_Band1,   Elevation, Hillsh
## min values  :                 0.194371,    0.000000,     26.000000,   72.217499,    0.0
## max values  :                15.945321,  106.665482,    140.000000,  212.632507,   32.4
```

This data set is a stack of 11 **rasterLayers** which correspond to mainly indices derived from a digital elevation model: elevation(**elevation**), terrain wetness index (**TWI**), altitude above channel network (**AACN**), terrain ruggedness index (**Terrain_Ruggedness_Index**), hillshading (**Hillshading**), incoming solar radiation (**Light_insolation**), mid-slope position (**Mid_Slope_Positon**), multi-resolution valley bottom flatness index (**MRVBF**), and slope gradient (**Slope**). Landsat 7 satelite spectral data, specifically band 1 (**Landsat_Band1** and normalised difference vegetation index (**NDVI**) are provided too, and give some indication of the spatial variation of vegetation patterns across the study area.

You may notice that there is a commonality between these **rasterlayers** in terms of their CRS, dimensions and resolution. When you have multiple **rasterlayers** as individual rasters, but they are common to each other in terms of resolution and extent, rather than working with each raster independently it is much more efficient to stack them all into a single object. The **stack** function from **raster** is ready-made for this, of which the **hunterCovariates_sub** stack is an output. This harmony is an ideal situation for DSM where there may often be instances where rasters from the some area under investigation may have different resolutions, extents and even CRSs. It these situations it is common to reproject and or resample to a common projection and resolution. The functions from the **raster** package which may be of use in these situations are **projectRaster** and **resample**.

While the example is a little contrived, it is useful to always determine whether or not the available covariates have complete coverage of the soil point data. This might be done with the following script which will produce a map like in Figure 3:

```
# plot raster
plot(hunterCovariates_sub[["Elevation"]],
main = "Hunter Valley elevation map with overlayed point locations")

## plot points
coordinates(HV_subsoilpH) <- ~x + y
plot(HV_subsoilpH, add = T)
```

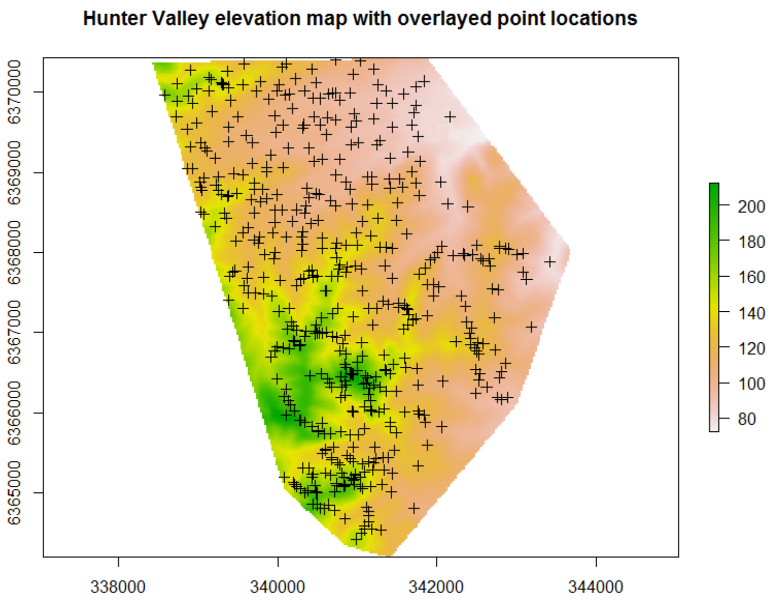

Figure 3: Hunter Valley elevation map with the soil point locations overlayed upon it.

With the soil point data and covariates prepared, it is time to perform the intersection between the soil observations and covariate layers using the script:

```
DSM_data <- extract(hunterCovariates_sub, HV_subsoilpH, sp = 1, method = "simple")
```

The `extract` function is quite useful. Essentially the function ingests the `rasterStack` object, together with the `SpatialPointsDataFrame` object `HV_subsoilpH`. The `sp` parameter set to 1 means that the extracted covariate data gets appended to the existing `SpatialPointsDataFrame` object. While the `method` object specifies the extraction method which in our case is “simple” which likened to get the covariate value nearest to the points i.e it is likened to “drilling down”.

A good practice is to then export the soil and covariate data intersect object to file for later use. First we convert the spatial object to a `dataframe`, then export as a comma separated text file.

```
DSM_data <- as.data.frame(DSM_data)
write.table(DSM_data, "hunterValley_SoilCovariates_pH.TXT",
col.names = T, row.names = FALSE, sep = ",")
```

2.1 Using rasters from file

In the previous example the rasters we wanted to use are available data from the `ithir` package. More generally we will have the raster data we need sitting on our computer or disk somewhere. The steps for intersecting the soil observation data with the covariates are the same as before, except we now need to specify the location where our raster covariate data is located. We need not even have to load in the rasters to memory, just point `R` to where they are, and then run the raster `extract` function. This utility is obviously a very handy feature when we are dealing with an inordinately large or large number of rasters. The work function we need is `list.files`. For example:

```
list.files(path = "C:/~~",
    pattern = "\\.tif$", full.names = TRUE)
```

```
##  [1] "C:/~~/AACN.tif"
##  [2] "C:/~~/Elevation.tif"
##  [3] "C:/~~/Hillshading.tif"
##  [4] "C:/~~/Landsat_Band1.tif"
##  [5] "C:/~~/Light_insolation.tif"
##  [6] "C:/~~/Mid_Slope_Positon.tif"
##  [7] "C:/~~/MRVBF.tif"
##  [8] "C:/~~/NDVI.tif"
##  [9] "C:/~~/Slope.tif"
## [10] "C:/~~/Terrain_Ruggedness_Index.tif"
## [11] "C:/~~/TWI.tif"
```

The parameter `path` is essentially the directory location where the raster files are sitting. If needed, we may also want to do recursive listing into directories that are within that path directory. We want `list.files()` to return all the files (in our case) that have the .tif extension. This criteria is set via the `pattern` parameter, such that $ at the end means that this is end of string, and but adding \\. ensures that you match only files with extension .tif, otherwise it may list (if they exist), files that end in .atif as an example. You may guess that any other type of pattern matching criteria could be used to suit your own specific data. The `full.names` logical parameter is just a question of whether we want to return the full pathway address of the raster file, in which case, we do.

All we then need to do is perform a raster stack of these individual rasters, then perform the intersection. This is really the handy feature where to perform the stack, we still need not require the loading of the rasters into the R memory — they are still on file!.

```
files <- list.files(path = "C:~~",
    pattern = "\\.tif$", full.names = TRUE)

# stack rasters
r1 <- raster(files[1])
for (i in 2:length(files)) {
    r1 <- stack(r1, files[i])
}
r1

## class       : RasterStack
## dimensions  : 249, 210, 52290, 11  (nrow, ncol, ncell, nlayers)
## resolution  : 25, 25  (x, y)
## extent      : 338422.3, 343672.3, 6364203, 6370428  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=56 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84 +to
## names       :       AACN,   Elevation, Hillshading, Landsat_Band1, Light_insolation, M
## min values  :    0.000000,   72.217499,    0.000677,     26.000000,      1236.662840,
## max values  :  106.665482,  212.632507,   32.440960,    140.000000,      1934.199950,
```

Note that the stacking of rasters can only be possible if they are all equivalent in terms of resolution and extent. If they are not you will find the other raster package functions `resample` and `projectRaster` as invaluable methods for harmonizing all your different raster layers. With the stacked rasters, we can perform the soil point data intersection as done previously.

```
DSM_data <- extract(r1, HV_subsoilpH, sp = 1, method = "simple")
```

3 Some exploratory data analysis

We will continue using the `DSM_data` object that was created in the previous section. As the data set was saved to file you will also find it in your working directory. Type `getwd()` in the console to indicate the specific file location. So lets read the file in using the `read.table` function:

```
hv.dat <- read.table("hunterValley_SoilCovariates_pH.TXT", sep = ",", header = T)
str(hv.dat)

## 'data.frame': 506 obs. of  15 variables:
##  $ id                     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x                      : num  340386 340345 340559 340483 340734 ...
##  $ y                      : num  6368690 6368491 6369168 6368740 6368964 ...
##  $ pH60_100cm             : num  4.47 5.42 6.26 8.03 8.86 7.28 4.95 5.61 5.39 3.44 ...
##  $ Terrain_Ruggedness_Index: num  1.34 1.42 1.64 1.04 1.27 ...
##  $ AACN                   : num  1.619 0.281 2.301 1.74 3.114 ...
##  $ Landsat_Band1          : int  57 47 59 52 62 53 47 52 53 63 ...
##  $ Elevation              : num  103.1 103.7 99.9 101.9 99.8 ...
##  $ Hillshading            : num  1.849 1.428 0.934 1.517 1.652 ...
##  $ Light_insolation       : num  1689 1701 1722 1688 1735 ...
##  $ Mid_Slope_Positon      : num  0.876 0.914 0.844 0.848 0.833 ...
##  $ MRVBF                          : num  3.85 3.31 3.66 3.92 3.89 ...
##  $ NDVI                           : num  -0.143 -0.386 -0.197 -0.14 -0.15 ...
##  $ TWI                            : num  17.5 18.2 18.8 18 17.8 ...
##  $ Slope                          : num  1.79 1.42 1.01 1.49 1.83 ...
```

Now lets firstly look at some of the summary statistics of soil pH.

```
round(summary(hv.dat$pH60_100cm), 1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.0     5.5     6.3     6.5     7.4     9.7
```

The observation that the mean and median are nearly equivalent indicates the distribution of this data does not deviate to much from normal. To assess this more formally, we can perform other analyses such as tests of skewness, kurtosis and normality. Here we need to use functions from the `fBasics` and `nortest` packages (If you do not have these already you should install them.)

```
library(fBasics)
library(nortest)
# skewness
sampleSKEW(hv.dat$pH60_100cm)
```

```
##      SKEW
## 0.1530612
```

```
# kurtosis
sampleKURT(hv.dat$pH60_100cm)
```

```
##     KURT
## 1.202168
```

Here we see that the data is slightly positively skewed. A formal test for normality of the Anderson-Darling Test statistic. There are others, so its worth a look at the help files associated with the `nortest` package.

```
ad.test(hv.dat$pH60_100cm)
```

```
##
##  Anderson-Darling normality test
##
## data:  hv.dat$pH60_100cm
## A = 4.0401, p-value = 4.594e-10
```

For this data to be normally distributed the p value should be > than 0.05. This is confirmed when we look at the histogram and qq-plot of this data in Figure 4.

```
par(mfrow = c(1, 2))
hist(hv.dat$pH60_100cm)
qqnorm(hv.dat$pH60_100cm, plot.it = TRUE, pch = 4, cex = 0.7)
qqline(hv.dat$pH60_100cm, col = "red", lwd = 2)
```

The histogram on Figure 4 shows something that you would expect from a

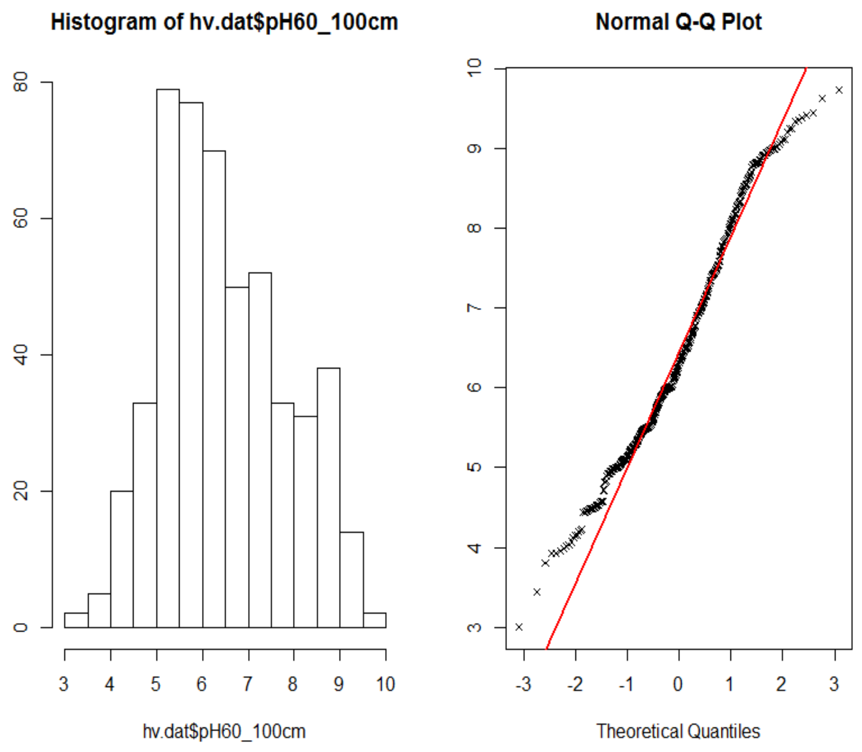

Figure 4: Histogram and qq-plot of soil pH in the 60-100cm depth interval.

normal distribution, but in practice fails the formal statistical test. Generally for fitting most statistical models, we need to assume our data is normally distributed. A way to make the data to be more normal is to transform it. Common transformations include the square root, logarithmic, or power transformations.

We could investigate other data transformations or even investigate the possibility of removing outliers or some such extraneous data, but generally we can be pretty satisfied with working with this data from a statistical viewpoint.

Another useful exploratory test is to visualize the data in its spatial context. Mapping the point locations with respect to the target variable by either altering the size or color of the marker gives a quick way to examine the target soil attribute spatial variability. Using the `ggplot2` package, we could create the plot as shown in Figure 5.

```
library(ggplot2)
ggplot(hv.dat, aes(x = x, y = y)) + geom_point(aes(size = hv.dat$pH60_100cm))
```

On Figure 5, there is a subtle north to south trend of low to high values, but generally it is difficult to make specific observations without consideration of the terrain and landscape these data are situated in.

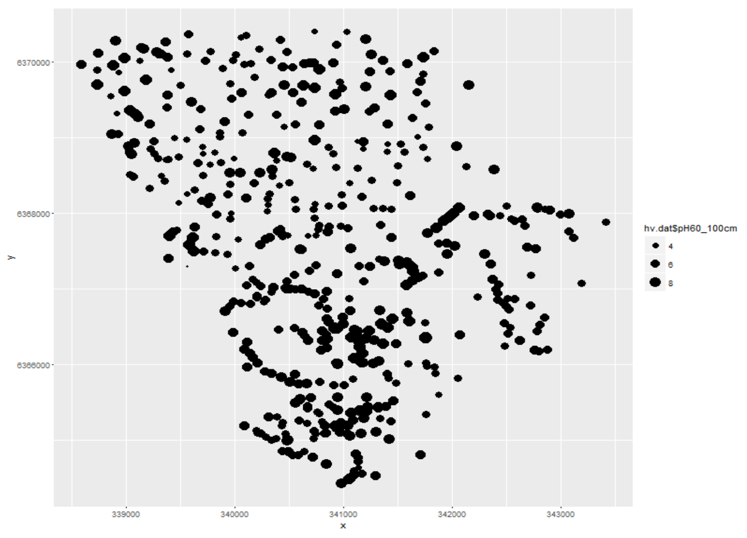

Figure 5: Spatial distribution of points in the Hunter Valley for the soil pH data at the 60-100cm depth interval.

Ultimately we are interested in making maps. So, as a first exercise and to get a clearer sense of the "spatial structure" of the data it is good to use some interpolation method to estimate soil pH values onto a regular grid of the study area extent. A couple of ways of doing this is the inverse distance weighted (IDW) interpolation and kriging.

For IDW predictions, the values at unvisited locations are calculated as a weighted average of the values available at the known points, where the weights are based only by distance from the interpolation location. Kriging is a similar distance weighted interpolation method based on values at observed locations, except it has an underlying model of the spatial variation of the data. This model is a variogram which describes the auto-correlation structure of the data as a function of distances. Kriging is usually superior to other means of interpolation because it provides an optimal interpolation estimate for a given coordinate location, as well as a prediction variance estimate. Kriging is very popular in soil science, and there are many variants of it. For further information and theoretical underpinnings of kriging or other associated geostatistical methods, with special emphasis for the soil sciences it is worth consulting Webster and Oliver (2001).

Functions for IDW interpolation and kriging are found in the `gstat` package. To initiate these interpolation methods, we first need to prepare a grid of points upon which the interpolation will be made. This can be done by extracting the coordinates from either of the 25m resolution rasters we have for the Hunter Valley. As will be seen later, this step can be made redundant because we can actually interpolate directly to raster. Nevertheless, to extract the pixel point coordinates from a raster we do the following using the `hunterCovariates_sub` raster stack. (Make sure both `raster` and `ithir` packages are loaded).

```r
data(hunterCovariates_sub)

tempD <- data.frame(cellNos = seq(1:ncell(hunterCovariates_sub)))
tempD$vals <- getValues(hunterCovariates_sub)
tempD <- tempD[complete.cases(tempD), ]
cellNos <- c(tempD$cellNos)
gXY <- data.frame(xyFromCell(hunterCovariates_sub, cellNos, spatial = FALSE))
```

The script above essentially gets the pixels which have values associated with them (discards all NA occurrences), and then uses the cell numbers to extract the associated spatial coordinate locations using the `xyFromCell` function. The result is saved in the `gXY` object.

Using the `idw` function from `gstat` we fit the formula as below. We need to specify the observed data, their spatial locations, and the spatial locations of the points we want to interpolate onto. The `idp` parameter allows you to specify the inverse distance weighting power. The default is 2, yet can be adjusted if you want to give more weighting to points closer to the interpolation point. As we can not evaluate the uncertainty of prediction with IDW, we can not really optimize this parameter.

```r
library(gstat)
IDW.pred <- idw(hv.dat$pH60_100cm ~ 1, locations = ~x + y,
data = hv.dat, newdata = gXY, idp = 2)
```

```
## [inverse distance weighted interpolation]
```

Plotting the resulting map (Figure 6) can be done using the following script.

```r
IDW.raster.p <- rasterFromXYZ(as.data.frame(IDW.pred[, 1:3]))
plot(IDW.raster.p)
```

For soil science it is more common to use kriging for the reasons that we are able to formally define the spatial relationships in our data and get an estimate of the prediction uncertainty. As mentioned before this is done using a variogram. Variograms measure the spatial auto-correlation of phenomena such as soil properties (Pringle and McBratney, 1999). The average variance between any pair of sampling points (calculated as the semi-variance) for a soil property S at any point of distance h apart can be estimated by the formula:

$$\gamma(h) = \frac{1}{2m(h)} \sum_{i=1}^{m(h)} \{s(x_i) - s(x_i + h)\}^2 \qquad (1)$$

where $\gamma(h)$ is the average semi-variance, m is the number of pairs of sampling points, s is the value of the attribute under investigation, x are the coordinates of the point, and h is the lag (separation distance of point pairs). Therefore, in accordance with the "law of geography", points closer together will show smaller semi-variance (higher correlation), whereas pairs of points farther away from each other will display larger semi-variance. A variogram is generated by

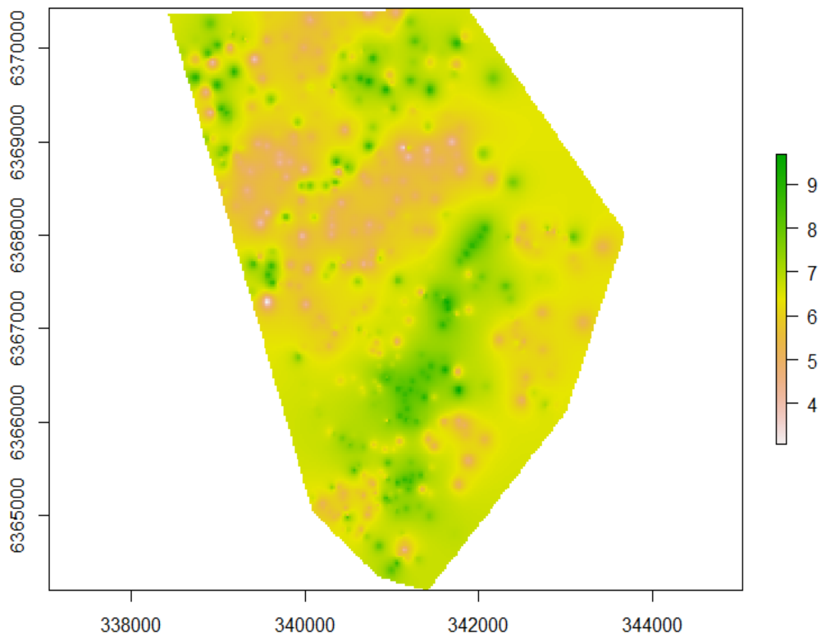

Figure 6: Map of soil ph (60-100cm) predicted using IDW

plotting the average semi-variance against the lag distance. Various models can be fitted to this empirical variogram where four of the more common ones are the linear model, the spherical model, the exponential model, and the Gaussian model. Once an appropriate variogram has been modeled it is then used for distance weighted interpolation (kriging) at unvisited locations.

First, we calculate the empirical variogram i.e calculate the semivariances of all point pairs in our data set. Then we fit a variogram model (in this case we will use a spherical model). To do this we need to make some initial estimates of this models parameters; namely, the nugget, sill, and range. The nugget is the very short-range error (effectively zero distance) which is often attributed to measurement errors. The sill is the limit of the variogram (effectively the total variance of the data). The range is the distance at which the data are no longer auto-correlated. Once we have made the first estimates of these parameters, we use the `fit.variogram` function for their optimization. The `width` parameter of the variogram function is the width of distance intervals into which data point pairs are grouped or binned for semi variance estimates as a function of distance. An automated way of estimating the variogram parameters is to use the `autofitVariogram` function from the `automap` package. For now we will stick with the `gstat` implementation.

```
vgm1 <- variogram(pH60_100cm ~ 1, ~x + y, hv.dat, width = 100, cutoff = 3000)
mod <- vgm(psill = var(hv.dat$pH60_100cm), "Exp", range = 3000, nugget = 0)
model_1 <- fit.variogram(vgm1, mod)
model_1
```

```
##   model     psill    range
## 1   Nug 0.3779266   0.0000
## 2   Exp 1.5337573 353.9732
```

The plot in Figure 7 shows both the empirical variogram together with the fitted variogram model line.

```
plot(vgm1, model = model_1)
```

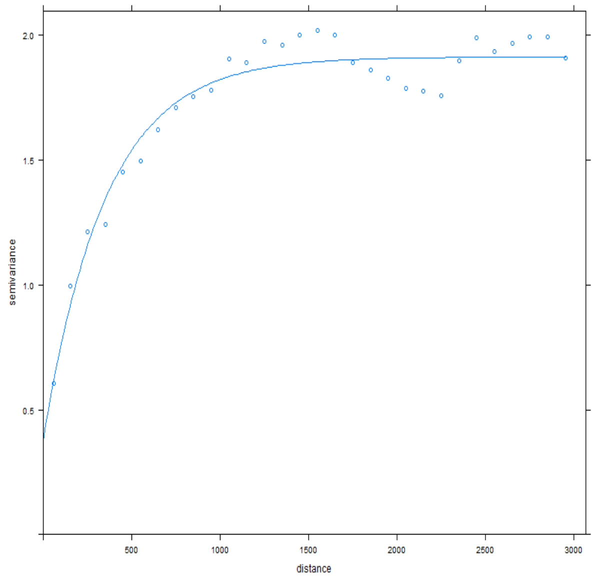

Figure 7: Empirical variogram and exponential variogram model of soil pH for the 60-100cm depth interval

The variogram is indicating there shows there is some spatial structure in the data up to around 500m. Using this fitted variogram model lets perform the kriging to make a map, but more importantly look at the variances associated with the predictions too. Here we use the `krige` function, which is not unlike using `idw` function, except that we have the variogram model parameters as additional information.

```
krig.pred <- krige(hv.dat$pH60_100cm ~ 1, locations = ~x + y,
data = hv.dat, newdata = gXY, model = model_1)
```

We can make the maps as we did before, but now we can also look at the variances of the predictions too (Figure 8).

```
par(mfrow = c(2, 1))
krig.raster.p <- rasterFromXYZ(as.data.frame(krig.pred[, 1:3]))
krig.raster.var <- rasterFromXYZ(as.data.frame(krig.pred[, c(1:2, 4)]))
plot(krig.raster.p, main = "ordinary kriging predictions")
plot(krig.raster.var, main = "ordinary kriging variance")
```

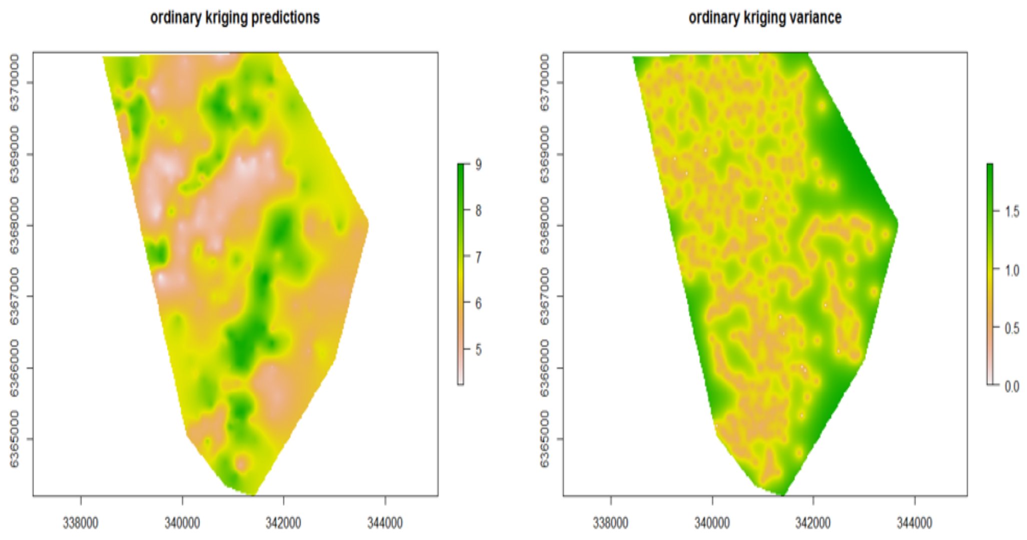

Figure 8: Kriging predictions and variances for log SOC (0-5cm).

Understanding the geostatisitcal properties of the target soil variable of interest is useful in its own right. However, it is also important to determine whether there is further spatial relationships in the data that can be modeled with environmental covariate information. Better still is to combine both spatial model approaches together (more of which will be discussed later on about this).

Ideally when we want to predict soil variables using covariate information is that there is a reasonable correlation between them. We can quickly assess these using the base `cor` function, for which we have used previously.

```
cor(hv.dat[, c("Terrain_Ruggedness_Index", "AACN", "Landsat_Band1", "Elevation",
    "Hillshading", "Light_insolation", "Mid_Slope_Positon", "MRVBF", "NDVI",
    "TWI", "Slope")], hv.dat[, "pH60_100cm"])
```

```
##                                  [,1]
## Terrain_Ruggedness_Index  0.146208599
## AACN                      0.194606533
## Landsat_Band1            -0.002769968
## Elevation                 0.148618926
## Hillshading               0.096981698
## Light_insolation         -0.046166532
## Mid_Slope_Positon        0.258329094
## MRVBF                    0.119674688
## NDVI                     0.162873990
## TWI                     -0.005330312
## Slope                    0.059593615
```

Progessing forwards later on, we will explore a range of models that will leverage what correlation there is between the target variable and the available covariates to create digital soil maps.

References

Arrouays, D., N. McKenzie, J. Hempel, A. Richer de Forges, and A. McBratney, eds.
2014. *GlobalSoilMap: Basis of the Global Spatial Soil Information System.* CRC Press.

Bishop, T. F. A., A. B. McBratney, and G. M. Laslett
1999. Modelling soil attribute depth functions with equal-area quadratic smoothing splines. *Geoderma*, 91:27–45.

Malone, B. P., A. B. McBratney, B. Minasny, and G. M. Laslett
2009. Mapping continuous depth functions of soil carbon storage and available water capacity. *Geoderma*, 154:138–152.

Minasny, B. and A. B. McBratney
2010. *Digital Soil Mapping: Bridging research, environmental application, and operation*, chapter 34. Methodologies for Global Soil mapping, Pp. 429–425. Dordrecht: Springer.

Ponce-Hernandez, R., F. H. C. Marriott, and P. H. T. Beckett
1986. An improved method for reconstructing a soil profile from analysis of a small number of samples. *Journal of Soil Science*, 37:455–467.

Pringle, M. J. and A. B. McBratney
1999. Estimating average and proportional variograms of soil properties and their potential use in precision agriculture. *Precision Agriculture*, 1:125–152.

Webster, R. and M. A. Oliver
2001. *Geostatistics for Environmental Scientists.* John Wiley and Sons Ltd, West Sussex, England.