Appendix B: The Data Sets

B.1 Data Set 1

B.1.1 General Description

The information in this section is taken from Greco (1999). The area selected for the study is located on the Sacramento River between river-mile 196, at Pine Creek Wildlife Area, and river-mile 219, near the southern tip of Woodson Bridge State Recreation Area. The primary criteria that led to the selection of this particular location were: (1) it was a river sector with minimal levee influence that permitted fluvial geomorphological processes and riparian forest successional processes to function, (2) it had adequate amounts and distribution of public lands needed for access to riparian forest stands, (3) historical aerial photographic data were available, (4) the location contained two gauging stations with historical flow data, and (5) there were adequate yellow-billed cuckoo survey data.

B.1.2 Land Cover Mapping from 1997 Aerial Photography

The land cover mapping scheme for this project is a modified version of the California Wildlife Habitat Relationships (CWHR) system's land classification (Mayer and Laudenslayer 1988). The three habitat variables mapped were: (1) land type, (2) vegetation height class, and (3) vegetation canopy cover class. A set of original aerial photographs of the study site was obtained from the California Department of Water Resources (CDWR) for the purpose of cartographic and photogrammetric interpretation of land cover types. The set consisted of six flight lines of true-color stereo-pair aerial photographs dated 14-May 1997 at a representative fraction (RF) scale of 1:12,000. Land cover mapping and interpretation was conducted using a minimum mapping unit (MMU) resolution of 20 meters.

The CWHR system defines tree habitats based on measurements of tree trunk diameter at breast-height (DBH) or crown width to determine the height class of tree habitats (Mayer and Laudenslayer 1988). Since these parameters are not easily measured from aerial photographs at a photographic scale of 1:12,000, the parameter "tree height" was selected as a measure of tree size. In this study, tree height classes were determined using visual estimates of relative tree height from a stereoscope and applying height strata classification breaks at 6 meters and 20 meters. Vegetation patches were estimated for height by delineating zones of similarly sized or nearly homogeneous stands. Only vegetation land types were classified for height categories, the remainder of the land types (e.g., water, developed areas, etc.) were assumed to be in the "low" category.

Canopy cover was visually estimated from the aerial photographs using a standard photo interpretation tree stocking density scale (Wenger 1984) that was photo-enlarged for use with 1:12,000 photographs. Boundaries of regions derived from photointerpretation were drawn on clear acetate sheets. These were scanned, imported into Arc/Info (ESRI, Redlands, CA), and converted to a polygon coverage. The polygon coverages were then transformed to geographic coordinates using control points stored in a Universal Transverse Mercator (UTM), Zone 10 projection. The individually interpreted maps (one from each photograph in each set) were then assembled into a composite database of the whole study reach using a manual edge matching method that minimized positional error between the vector pairs. The composite database is included in Data Set 1 as the polygon shapefile *landcover.shp*. The coordinate system is UTM Zone 10, WGS 84. In addition to the *FID* and *Shape* fields, the attribute table contains the following data fields:

Area: The polygon area in m^2

Perimeter: The polygon perimeter in m

VegType: Land cover type according to the following code (the bold types are those that may serve as yellow-billed cuckoo habitat): annual grassland (*a*), cropland (*c*), urban/developed (*d*), **freshwater emergent wetland (*f*),** gravel bar (*g*), **lacustrine (non-flowing) (*l*),** orchard (*o*), riverine (flowing) (*r*), **riparian (*v*)**, valley oak woodland (*w*)

HeightClass: Tree height in three categories: low (*l*), less than 6m; medium (*m*), 6–20m; and high (*h*), greater than 20m.

CoverClass: Cover class in three categories: sparse (*s*), 10–24 percent; open, 25–39 percent; moderate (*m*), 40–59 percent; dense (*d*), 60–100 percent; open (0), open water.

Data were ground field verified as described in Greco (1999).

B.1.3 Habitat Patches

The polygon shapefile *habitatpatches.shp* disaggregates the land cover data into patches as defined in the CWHR model (see Section 7.3). The file was created from the file *landcover.shp* through a series of GIS operations in ArcGIS that replicated the original operations carried out by Greco (1999). Only potential cuckoo habitat patches are included in the shapefile. These consist of polygons of land cover class freshwater emergent wetland (*f*), lacustrine (non-flowing) (*l*), or riparian (*v*) that are either within 100m of the river or directly adjacent to a polygon within 100m of the river. The steps used to create the file were as follows.

- **Step 1.** Add *landcover.shp*.
- **Step 2.** Create a shapefile as a 100m buffer of water. Select by location all polygons in *landcover.shp* that intersect this buffer. Then select all polygons in *landcover.shp* that touch the boundary of a selected polygon. Next select *VegType* = “f,” “l,” or “v.” Then export this selection as the shapefile *riprval100m.shp*.
- **Step 3**. Use the ArcGIS *Dissolve* tool to dissolve the shapefile created in Step 2 on *VegType*.
- **Step 4.** Use the ArcGIS *Explode* tool to separate polygons into distinct entities. This does not create a new file.
- **Step 5.** Calculate area and perimeter of polygons in the shapefile.
- **Step 6.** Define a data field PatchID. Save this file as *habitatpatches.shp*.

One of the variables in the CWHR model is *patch width*. This was estimated in ArcGIS by converting the polygon shapefile *habitatpatches.shp* to raster and using the Raster Calculator command *zonalgeometry* to compute the thickness of each patch. The patch width was taken to be twice the thickness. This grid was then converted to vector and joined to the attribute table of *habitatpatches.shp* via an attribute join.

B.1.4 The Creation of the Full Set of Explanatory Variables

The shapefiles *landcover.shp* and *habitatpatches.shp* are not strictly necessary for the analysis of the data set and are included to aid in data exploration. The full set of explanatory variables is contained in the shapefile *set1data.shp*. This file was constructed in ArcGIS from the files *landcover.shp*, *habitatpatches.shp*, and a polygon shapefile named *floodplnage.shp*. This last file contained floodplain age estimated by the method described by Greco et al. (2007). The age categories are (in years) 10, 19, 31, 45, 59, 93, 101, and 102. Each category except the last represents the estimated number of years, based on interpretation of historical aerial images and maps, since the patch was last under water. The last category, 102, indicates patches that have not been under water in the last 101 years, which is as far back in time as accurate historical maps exist.

The shapefile *set1data.shp* was created as follows.

- **Step 1.** Union *landcover.shp* and *habitatpatches.shp.*
- **Step 2.** Eliminate redundant data fields and export the shapefile.
- **Step 3.** Intersect *floodplnage.shp* with the shapefile. Eliminate redundant data fields.
- **Step 4.** Save resulting file as *set1data.shp.*

The polygons in the file *set1data.shp* are thus subdivided according to habitat patch (which is defined by vegetation type), age class, and height class. Each polygon of *set1data.shp* is characterized by a unique combination of these three factors. The attribute data fields of *set1data.shp* include *PatchID*, *AgeID,* and *HtClID*, which are the ID number of the polygon determining each of these quantities, *PatchArea, AgeArea,* and *SzClArea*, which are the areas of each of these polygons, *PatchWidth, VegType, Age*, and *HeightClass*, which are the variables defining each polygon.

It happened that two separate habitat patches were given the same *PatchID* value (100) in the original data set. As luck would have it, each of these contained a cuckoo observation point. Based on the values of *HtClID*, one of these patches was reassigned the *PatchID* value 400.

B.1.5 Yellow-Billed Cuckoo Observation Points

The file *obspts.csv* contains the yellow-billed cuckoo observation data. This is an aggregation of observations collected during eleven years over a period between 1978 and 1992 (Gaines, 1974; Gaines and Laymon, 1984; Laymon and Halterman, 1987; Halterman, 1991, Greco, 2002). The survey includes a total of 21 locations within the study area. The five surveys each followed a similar protocol in which the presence or absence of the yellow-billed cuckoo is tested by visiting the site, playing a tape recording of the bird's *kowlp* call, and observing whether or not there is a response (Gaines, 1974). The standard procedure (Greco, 1999, p. 232) is to play the call five to ten times at each observation point, although the exact procedure varied somewhat. The call is audible for a distance of about 300 meters (Laymon and Halterman, 1987). Failure to elicit a response is taken to imply that no cuckoos are present at that location. The response may be simply a bird call and may not include an actual sighting. The sites are generally visited by boat and are often impenetrable, in which case the call is played from the boat. The original coordinates of two of the observation points (point 4 and point 19) in the data set were located in the river itself. These were taken to be the location of the boat at the time the call was played. They were adjusted to place the location in the nearest point on land.

B.1.6 The Data Files

The data describing Data Set 1 are housed in three entities: two ESRI format shapefiles and one comma separated variable file. The ESRI shapefiles actually consist of three disk files each the description of the contents of the files is given in Section 7.2. Assuming that the `setwd()` function has been used to set the working directory to the data directory, the data are loaded using the following R statements.

The data from Data Set 1 are housed in the *Set1* subdirectory of the *Data* directory. The shapefile consisting of the files *set1data.shp*, *set1data.shx*, set1data.*dbf* contains the data describing the explanatory variables. The statements to load this data set as a spatial features file are:

```
> library(sf)
> data.Set1.sf <- st_read("set1\\set1data.shp")
> st_crs(data.Set1.sf) <- "+proj=utm +zone=10 +ellps=WGS84"
```

To convert his to a `SpatialPolygonsDataFrame` object, we use the coercion process.

```
> library(maptools)
> data.Set1.sp <- as(data.Set1.sf, "Spatial")
```

The comma separated variable file *obspts.csv* contains the data associated with the yellow-billed cuckoo observations. The statements to load this data set, convert it to a `SpatialPointsDataFrame` object, and assign coordinates are:

```
> library(maptools)
> data.Set1.obs <- read.csv("set1\\obspts.csv", header = TRUE)
> coordinates(data.Set1.obs) <- c("Easting", "Northing")
> proj4string(data.Set1.obs) <- CRS("+proj=utm +zone=10 +ellps=WGS84")
```

The data may also be converted from a data frame into an `sf` object.

```
> library(sf)
> data.Set1.obs.sf <- st_as_sf(data.Set1.obs, coords = c("Easting",
+   "Northing"))
```

Two other data sets are provided. The shapefile consisting of the files *habitatpatches.shp*, *habitatpatches.shx*, and *habitatpatches.dbf* contains the habitat patches data. The statements to load this data set are:

```
> library(maptools)
> data.Set1.patches.sf <- st_read("set1\\habitatpatches.shp")
> st_crs(data.Set1.patches.sf) <- "+proj=utm +zone=10 +ellps=WGS84"
> data.Set1.patches.sp <- as(data.Set1.patches.sf, "Spatial")
```

The shapefile consisting of the files *landcover.shp*, *landcover.shx*, and *landcover.dbf* contains the land cover class data. The statements to load this data set are:

```
> library(maptools)
> data.Set1.landcover.sf <- st_read("set1\\landcover.shp")
> st_crs(data.Set1.landcover.sf) <- "+proj=utm +zone=10 +ellps=WGS84"
> data.Set1.landcover.sp <- as(data.Set1.landcover.sf, "Spatial")
```

B.2 Data Set 2

Data Set 2 is based on 4,101 plots from a historic dataset, the Vegetation Type Map (VTM) survey of California (Wieslander, 1935). This survey was conducted in the 1920s and 1930s. Field crews gathered data from more than 8,000 plots in the coastal ranges from San Francisco bay to the Mexican border and a large part of the Sierra Nevada. Allen and coworkers entered approximately 4,300 VTM records (all the plots featuring a *Quercus* species) into a database and analyzed them to construct a classification system for California's oak woodlands (Allen et al. 1991). Evett (1994, p. 27) added geographic coordinates and estimated climatic data to the database and built models of the environmental niche for six oak species using the GLM-based direct gradient analysis approach of Austin et al. (1990).

The VTM survey plots were 0.081 hectare rectangles chosen to be representative of the vegetation map units being delineated in the field on topographic maps (Wieslander, 1935). Vegetation information consisted of the number of stems per diameter class of overstory tree species and the percent cover of each understory plant species. Original environmental information included plot location, elevation, slope, aspect, soil surface texture, and parent material (Evett, 1994, p. 22). Temperature, length of growing season, and evapotranspiration data for each plot were derived by Evett (1994) from a series of county level isoline maps drawn by C. R. Elford, the California state climatologist, in the 1960s and 1970s. Vayssières et al. (2000) derived precipitation data from an isohyetal map of thirty-year annual average precipitation (1950 to 1980) drawn from approximately 4,100 stations by the California Department of Water Resources. Average available soil water capacity was computed from the State Soil Geographic (STATSGO) GIS soils database for California. Both precipitation and soil map data were at 1/250,000 scale. Values for individual plots were interpolated and/or extracted using the GIS program Arc/Info (ESRI, Redlands, CA). Potential cloudless solar radiation on a tilted surface (in MJ/m^2) was computed as a function of latitude, altitude, slope, and aspect using a FORTRAN program written by T. Rumsey (personal communication).

The data from Data Set 2 are housed in the *Set2* subdirectory of the *Data* directory. The data describing Data Set 2 are housed in the comma separated variable file *set2data.csv*. Assuming that the `setwd()` function has been used to set the working directory to the data directory, the data are loaded using the following R statement.

```
> data.Set2 <- read.csv("set2\\set2data.csv", header = TRUE)
```

A complete list of the quantities contained in Data Set 2 and the symbols that represent them is given in Table 7.2.

Because the locations in Data Set 2 span two UTM zones, longitude and latitude are generally used to represent spatial coordinates.

B.3 Data Set 3

A total of 16 fields, each planted to rice (*Oryza sativa* L.) were sampled during the course of three growing seasons, 2002–03, 2003–04, and 2004–05. The fields were located in three separate regions in northeastern Uruguay, latitude 32.8° to 33.8° S, longitude 53.7° to 54.5° W. A total of 323 data locations were sampled, 125 in 2002–03, 108 in 2003–04, and 100 in 2004–05 (Figure 1.8). Sample sites were selected by eye, with the intent of obtaining a data set that appropriately represented field variability. Field size varied between 30 to 50 ha. On average there was approximately one sample site per 2.5 ha. Sample sites were georeferenced with a hand held Garmin global positioning system (Garmin International, Oalthe, KS). In fields that were sampled in multiple years, sample sites were at the same location in each year. In addition to longitude, latitude, and a number identifying the field, a total of 19 attribute data items were recorded at each point (Table 7.3). Four of the data fields, emergence (*Emer*), weed level (*Weeds*), level of weed control (*Cont*), and irrigation effectiveness (*Irrig*) were scored on an ordinal scale of 1 to 5 based on expert judgment. Planting date (*DPL*) was recorded on a scale of 1 to 5 based on half-month increments from 1 October, so that 1 represented 1 October through 15 October, 2 represented 16 October through 31 October, etc. The approximate date of 50 percent flowering (*D50*) was recorded as the date the field was visited estimated to be that closest to the true date of 50 percent flowering. Prior to planting, at each sample site in each field, four soil cores (0–30 cm depth) were extracted from a circular region of about 8 m radius such that one core lay in each of the four quadrants. Sand (*Sand*), silt (*Silt*), and clay (*Clay*) content were measured. Soil pH (*pH*), organic matter (*Corg*), phosphorus (Bray) (*SoilP*), and potassium (*SoilK*) were determined at the Instituto Nacional de Investigación Agropecuaria (INIA) La Estanzulea Soils Laboratory.

A sample plot of size 2.5 × 3m was hand harvested at each location and threshed in an experimental combine. Yields were obtained by hand harvesting, standardized to 14 percent moisture content, and these standardized yields (*Yield*) and harvest moisture were recorded, as well as variety (*Var*). Climatic data were recorded at the INIA El Paso de la Laguna Rice Experiment Station near Treinta y Tres. Daily average temperature, rainfall and total hours of sun were summarized by period, where each month is divided into three periods of length approximately ten days. Period 1 contains the first third of October, Period 2 the second, and so on until Period 21, which contains the last third of April.

The data from Data Set 3 are housed in the *Set3* subdirectory of the *Data* directory. The data describing Data Set 3 are housed in the comma separated variable file *set3data.csv*. Assuming that the `setwd()` function has been used to set the working directory to the data directory, the data are loaded using the following R statement.

```
> data.Set3 <- read.csv("set2\\set3data.csv", header = TRUE)
```

B.4 Data Set 4

B.4.1 General Description

The site at which the study took place consists of two commercial fields located in Winters, CA, latitude 38°32′ N, longitude 121°58′ W. We identify the fields as Fields 1 and 2. They are 32 and 31 ha in area, respectively. Soils are predominantly alluvial clay loams, silty clay loams, and silty clays. Mean annual precipitation is 55.0 cm, almost all of which occurs during the winter. The fields, which had been laser leveled, were graded to a uniform slope for furrow irrigation. Spring wheat (*Triticum aestivum* L. cv. "Express") was drill seeded in December, 1995 into raised beds with furrows spaced 1.52 m apart. The furrows provided surface drainage (the fields do not have any other drainage system installed) during periods of heavy rain that occurred in January, 1996 and were used for a single irrigation in April, 1996 at anthesis. Normal commercial practices, including pest management and supplemental fertilization, were followed in producing the crop. Precipitation during the winter of 1995–96 was 82.2 cm, or 150 percent of normal. The wheat crop was harvested in June 1996.

Aerial photographs were taken on December 16, 1995, March 8, 1996, and May 4, 1996 from an altitude of approximately 1000 m using 70 mm Kodak Aerochrome II 2443 color infrared film. Positive images were scanned using an Agfa Arcus II scanner at 600 dpi. This corresponded to a resolution of approximately 1.8 m. Point sampling was carried out in 1995 on a 61 m square grid in each field. There were 86 and 78 points sampled in Fields 1 and 2, respectively. At each sample point, 25 soil cores (0–15 cm depth) were extracted prior to planting from the tops of the beds in a 25 m^2 area and composited. Sand, silt, and clay content were measured. Soil pH, nitrogen, phosphorus, potassium, and organic carbon levels were determined using standard methods of the University of California Division of Agriculture and Natural Resources Analytical Laboratory (Anonymous, 1998). At anthesis, visual determinations were made at each grid point of stand density and weed and disease infestation intensity, each of which were rated on a scale of 1 to 5. Locations of sample points were determined using a Trimble ProXL® GPS (Trimble Navigation, Sunnyvale, CA), which was corrected by post-processing.

In the second year, both fields were planted to tomato (*Solanum lycopersicum* L.). In the third year, Field 1 was planted to bean (*Phaseolus vulgaris* L.) and Field 2 was planted to sunflower (*Helianthus annuus* L.), and in the fourth year Field 1 was planted to sunflower and Field 2 was planted to corn (*Zea mays* L.). No soil or plant sampling was carried out after the first year. Grain crops were harvested with a combine equipped with an Ag Leader®/GPS yield mapping system (Ag Leader Technology, Ames, Iowa), and tomatoes were harvested with an experimental tomato yield monitor (Pelletier and Upadhyaya, 1999). Yield data were collected once per second and in some years it was not possible to get full coverage.

The data from Data Set 4 are housed in the *Set4* subdirectory of the *Data* directory. Assuming that the `setwd()` function has been used to set the working directory to the data directory, the data are loaded using the following R statements.

B.4.2 Weather Data

Weather data recorded daily at the University of California, Davis, about 20 km from the fields, is housed in the comma separated variable file *dailyweatherdata.csv*. The file is loaded using the following statement.

```
> weather.data <- read.csv("Set4\\dailyweatherdata.csv", header = TRUE)
```

B.4.3 Field Data

The following describes the loading of data from Field 1. The statements for the Field 2 data are identical except for the obvious substitution of "2" for "1" in the appropriate places.

The data collected manually at the 86 sample sites in Field 1 are housed in the comma separated variable file *set4.196sample.csv*. This file is loaded using the following statement.

```
data.Set4.1 <- read.csv("Set4\\set4.196sample.csv", header = TRUE)
```

"Raw" yield monitor data are housed in comma separated variable files *set4.196wheatyield.csv*, *set4.197tomatoyield.csv*, *set4.198beanyield.csv*, and *set4.199sunfloweryield.csv*. The 1996 data are loaded using the following statement.

```
> data.Set4.1Yield96raw <-
+    read.csv("Set4\\Set4.196wheatyield.csv", header = TRUE)
```

Analogous statements are used for the other files.

Georeferenced image data for remotely sensed images acquired in December 1995, March 1996, and May 1996 are stored in the following file pairs: *set4.11295.tif* and *set4.11295.tfw*; *set4.10396.tif* and *set4.10396.tfw*; and *set4.10596.tif* and *set4.10596.tfw*, respectively. The data can be loaded using the function `readGDAL()` or `raster()`. The following statement are used to load these data using `readGDAL()`.

```
> library(rgdal)
> data.4.1.Dec <- readGDAL("set4\\set4.11295.tif")
> data.4.1.Mar <- readGDAL("set4\\Set4.10396.tif")
> data.4.1.May <- readGDAL("set4\\Set4.10596.tif")
```

The following statements may be used to load the data as a `raster brick` object.

```
> library(raster)
> data.4.1.Dec <- brick("set4\\set4.11295.tif")
> data.4.1.Mar <- brick("set4\\Set4.10396.tif")
> data.4.1.May <- brick("set4\\Set4.10596.tif")
```

In Section 6.4.2, the contents of the "raw" file *set4.196wheatyield.csv* are cleaned, and the result is stored in the comma separated variable file *set4.1yld96cleaned.csv* in the *created* subdirectory. Data from this file are loaded using the following statement:

```
> data.Yield4.1idw <-
+    read.csv("created\\set4.1yld96ptsidw.csv", header = TRUE)
```

Field 2 of Data Set 4 contains a data set of high density apparent electrical conductivity measurements. Data from these measurements are stored in the comma separated variable file *Set4.2EC.csv*. Data from this file are loaded using the following statement.

```
> data.Set4.2EC <- read.csv("Set4\\Set4.2EC.csv", header = TRUE)
```

Appendix C: An R Thesaurus

R is a functional programming language, and this provides the user with both its greatest power and its greatest challenge. It is powerful because when you want to carry out some task, there is a very good chance that someone else has written a function that will carry out this task and has left the function for you to use. It is challenging because you have to find the function and learn how to use it. Then you have to remember it. This is particularly a problem for people who don't use the program on a regular basis. When I was first learning R, I found it very useful to keep a record of tasks I had performed and the functions I had used to perform them. This motivated me to create such a list of tasks and functions in this book. A thesaurus is a book that you use when you have an idea and you want to find a word to fit it, and this appendix provides something you can use when you have a task and you want to find a function to perform it. For that reason, I call this an "R Thesaurus." The thesaurus is divided into major categories of operations and subcategories within these categories. The task and function are listed, along with the package in which the function is found. If no package is listed, then the function is in the base package. The R file and the line number within the file identify one location of this function. If the function is discussed in the text, then this is the reference implementation; otherwise, it is generally the first implementation. There may be other implementations of the function in other files as well. To find these, a good text search app is useful. The one I like is Agent Ransack (https://www.mythicsoft.com/agentransack/).

C.1 Data Input and Output

Operation	Function	Package	File	Line
Read an attribute data file	`read.csv()`		2.4.1	2
Read a GeoTiff file	`readGDAL()`	`Rgdal`	1.1	8
	`brick()`	`raster`	2.4.3	81
Read an ESRI shapefile into an `sf` object	`st_read()`	`sf`	2.4.3	50
Write an `sf` object to an ESRI shapefile	`st_write()`	`sf`	2.4.3	46

C.2 Working with Objects

C.2.1 Attribute Data Objects and General Object Functions

Operation	Function	Package	File	Line
Convert a numeric object to a factor	`as.factor()`		2.4.1	18
Create an ordered factor with labels	`ordered()`		7.2	59
Display the names of the data fields of a data frame	`names()`		2.4.1	7

(*Continued*)

Operation	Function	Package	File	Line
Display only the first records of a data frame	`head()`		4.5.1	59
Display only the last records of a data frame	`tail()`		4.5.1	60
Display the class of an object	`class()`		2.2	5
Display the structure of an object	`str()`		2.4.1	3
Display the contents of a slot of and S4 object	`slot()`		2.4.3	62
Display only one element of an object with a long `str()` output	`str()`		5.3.4	130–131
Remove data records with missing values	`is.na()`		10.1	23
Construct a list	`list()`		2.4.1	21
Construct a function	`function()`		2.5	16
Convert a number to characters	`as.character()`		3.6.3	5
Paste character strings together	`paste()`		6.2.3	7
Convert a list to a vector	`unlist()`		2.4.3	68
Display the length of a vector	`length()`		2.4.3	25
Create a matrix	`matrix()`		2.2	54
Bind together vectors to form a matrix	`cbind()`		2.2	60
Assign names to rows and/or columns of a matrix	`rownames()` `colnames()`			
Construct a data frame	`data.frame`		2.4.1	28
Reference a data frame	`with()`		5.3.4	36
Add fields to a data frame	`data.frame()`		3.2.1	18–19
Select a subset of members of a data frame	`subset()`		7.5	129
Create a contingency table	`table()`		12.6.2	67
Generate a sequence of numbers	`seq()`		2.4.3	103
Coerce a sequence into a matrix	`as.matrix()`		10.1	10
Concatenate the columns of a matrix into a vector	`stack()`		3.2.1	196
List all of the unique values of an array	`unique()`		6.2.3	36
Center and scale the values of an array	`scale()`		7.2	212
Arrange the elements of an array in order of value of elements of another array	`order()`		6.2.3	59
Create an array of factors based on characters	`factor(), as.character()`		1.3.1	61
Coerce an object into a factor	`as.factor()`		2.4.1	18
Coerce an object into an integer	`as.integer()`		2.5	5
Associate labels with factors	`factor()`		8.4.2	101
Display the levels of a factor	`levels()`		7.2	57
Determine the array number of the smallest element in an array	`which.min()`		5.3.1	28
Join two data frames by merging elements with matching attribute values	`merge()`		7.2	167
Remove duplicate rows from a data frame	`remove.dup.rows()`	`cwhmisc`	9.3.4	76
Execute statements based on a condition	`if()` `ifelse()`		2.3.1 3.2.2	11 15

C.2.2 Spatial Data Objects

Operation	Function	Package	File	Line
Convert a data frame to an `sf` object	`st_as_sf()`	`sf`	2.4.3	9
Covert an `sf` object to an `sp` object	`as()`	`sp`	2.4.3	59
Covert an `sp` object to an `sf` object	`st_as_sf()`	`sf`	3.6.2	5,7
Convert a data frame to a `SpatialPointsDataFrame`	`coordinates()`	`sp`	2.4.3	73
Assign a projection to an `sf` object	`st_crs()`	`sf`	2.4.3	16
Assign a projection to an `sp` object	`proj4string()`	`sp`	2.4.3	74
Transform from one projection to another	`spTransform()`	`rdgal`	12.6.2	19
Create a grid of points	`expand.grid()`		2.4.3	103
Convert a `SpatialPointsDataFrame` to a `SpatialGridDataFrame`	`gridded()`	`sp`	4.6.1	35
Convert a band from an image to a `SpatialPointsDataFrame`	Several	`maptools`	1.1	77–82
Create an `sf` polygon object	Several	`sf`	2.4.3	37–43
Extract the coordinates of the label points of a `SpatialPolygons` object	`lapply()`		2.4.3	67
Extract the `ID` values of a `SpatialPolygons` object	`lapply()`		2.4.3 6.4.2	66 73
Digitize a region on the screen	`locator()`	`sp`	7.3	123
Select those points in a `SpatialPointsDataFrame` lying inside a `SpatialPolygons` object	`over()`	`sp`	5.3.4 7.5	121 43
Compute the means of an attribute value over points lying within a group of polygons	`over()`	`sp`	6.4.2 11.5.1	77 46

Resample from one grid to another	`raster()`
	`over()`
		`raster`
		`sp`
			6.4.3
			6.4.2
				17
				77

Do a "polygon containing point" operation to create a data frame of attributes of polygons containing a set of points	`over()`	`sp`	7.2	120
Clip an interpolated grid to an irregular boundary	`over()`	`sp`	7.3	141
Convert a SpatialPoints object into a SpatialGrid object	`gridded()`	`sp`	4.6.1	35
Construct a regular lattice of polygons	`GridTopology()`, `as()`	`sp`	5.3.2	32
Create a set of Thiessen polygons using `spatstat` functions	Several	`spatstat`	3.6.1	41–44
Verify that attribute values match spatial locations correctly	`Several`		3.6.2	4–18
Create buffers around points	`disc()`	`spatstat`	6.4.2	36
Construct a neighbor list for a lattice	`cell2nb()`	`spdep`	3.2.1	9
Construct a neighbor list for `SpatialPolygons` or `SpatialPolygonsDataFrame` object	`poly2nb()`	`spdep`	3.6.3	4
Assign weights to a spatial weights matrix	`nb2listw()`	`spdep`	3.5.3	11
Assign weights to spatial weights matrix based on distance	`dnearneigh()`	`spdep`	3.6.1	13
Assign weights to spatial weights matrix based on number of neighbors	`knearneigh()`	`spdep`	3.6	23
Determine the cardinality (number of neighbors) of a point set	`card()`	`spdep`	14.5.2	13
Generate an autocorrelated data set	`invIrM()`	`spdep`	3.2.1	11
Bind two `SpatialPointsDataFrame` objects	`spRbind()`	`sp`	5.2.1	38
Crop a `raster` object	`crop()`	`raster`	7.4	72
Extract cell values from a raster layer	`extract()`	`raster`	5.7	38
Compute the slope and aspect of a raster layer	`slopeAspect()`	`raster`	7.4	73

C.3 Graphics

Graphics functions are discussed extensively in Section 2.6. The discussion, however, references figures in other chapters. The code to produce these figures is in the section containing the figure, rather than in Section 2.6. Also note that `ggplot()` can do many quick plotting operations much more easily.

C.3.1 Traditional Graphics, Attribute Data

Section 2.6.1 contains a discussion of traditional graphics for attribute data.

Operation	Function	Package	File	Line
Create a scatterplot	`plot()`		3.5.1	59
Create a line plot	`plot()`		3.2.1	46
Create a histogram	`hist()`		3.3	13
Create a histogram of two data sets	`hist(), plot()`		10.1	36
Create a hexagonal bin plot	`hexbin()`	`hexbin`	7.3	244
Create a boxplot	`boxplot()`		7.2	193
Create a barplot	`barplot()`		7.3	215
Create an aligned dot plot	`matplot()` `plot()`		7.2	218
Identify individual points in a scatterplot	`identify()`		6.3.3	57
Insert italic script	`expression()`		3.2.1	47
Insert a subscript	`expression()`		4.6.1	23
Make text bold	`expression()`		4.6.1	23
Insert a Greek character	`expression()`		3.5.1	59
Place a bar over a character	`expression()`		3.5.1	63
Insert a degree symbol	`expression()`		7.4	101
Examples of the use of expression() for complex formulas	`expression()`		3.2.1 4.6.1	53 44
Draw an arrow	`arrows()`		4.6.1	22
Insert a line of text	`text()`		3.2.1	51
Control the position of the text	`text()`		3.2.1	51
Define *x* and *y* limits of a function	`plot()`		4.6.1	41
Draw contours of a surface	`contour()`		12.6.1	58
Control plotting of axes	`axis()`		7.4	106
Plot an empty screen	`plot()`		7.1	1
Control the size of axis labels	`plot()`		3.2.1	45
Control the size of the margins	`par()`		3.2.1	45
Construct a perspective plot	`persp()`		3.2.1	76
Construct a circle	`symbols()`		3.5.2	41
Draw an ellipse	`ellipse()`	`car`	3.5.2	43

C.3.2 Traditional Graphics, Spatial Data

Section 2.6.2 contains a discussion of traditional graphics for spatial data.

Operation	Function	Package	File	Line
Define a gray scale	`grey()`		3.5.3	39
Display a GeoTiff file	`image()`		5.3.4	50
Display a `SpatialPointsDataFrame` over a GeoTiff image	`plot()`	`maptools`	1.1	17
Construct a bubble plot	`plot()`	`maptools`	3.2.1	163
Select colors for an image	`terrain.colors()`	`RColorBrewer`	1.1	26
Plot a map showing geographic coordinates on the axes	`title()`	`maptools`	1.1	21
Draw a map using data from `stateMapEnv`	Several	`maps, maptools`	1.3.1 7.3	5 25
Draw a map using data from `world`	Several	`maps, maptools`	1.3.3	4
Draw a scale bar on a map	`SpatialPolygonsRescale()`	`maptools`	1.3.1 7.3	37 39
Draw a north arrow on a map	`SpatialPolygonsRescale()`	`maptools`	1.3.1	28
Draw a small map inside a larger map	Several	`maptools`	1.3.1	40
Plot the value of an attribute by spatial location	`text(),`	`maptools`	4.5.3	23
Create a star plot map	`stars()`		7.5	172

C.3.3 Trellis Graphics, Attribute Data

Section 2.6.3 contains a discussion of trellis graphics for both attribute and spatial data.

Operation	Function	Package	File	Line
Create scatter or line plots	`xyplot()`	`lattice`	4.5.1 7.4	63 174
Create a scatterplot matrix	`splom()`	`lattice`	7.3	119
Print a `lattice` graph in gray scale	`trellis.device()`	`lattice`	3.5.3	40
Create a boxplot	`bwplot()`	`lattice`	7.4	27
Create a dotplot	`dotplot()`	`lattice`	8.3	22
Construct a trellis plot with two grouping variables	`Any trellis graphics function`	`lattice`	7.4	174
Adjust the parameters of a trellis graphics function	`Any trellis graphics function`	`lattice`	7.3	118

C.3.4 Trellis Graphics, Spatial Data

Section 2.6.3 contains a discussion of trellis graphics for both attribute and spatial data.

Operation	Function	Package	File	Line
Plot an interpolated surface	`spplot()`	`sp`	1.1	59
Plot a `SpatialPointsDataFrame` over an interpolated surface	`spplot()`	`sp`	1.2	20–23
Create a north arrow	`spplot()`	`sp`	1.3.1	65
Create a scale bar	`list()`	`sp`	1.3.1	67
Control the placement of the legend	`spplot()`	`sp`	15.2	135
Plot a thematic map of factor data (polygons)	`spplot()`	`sp`	7.2	65
Create a color map	`spplot()`	`sp`	1.3.1	80, 101
Create multiple panel plots	`spplot()`	`sp`	3.5.3	41
Overlay a point map on top of a polygon map.	`spplot()`	`sp`	7.2	65
Plot in color	`colorRampPalette()`	`sp`	3.5.3	48

C.4 Computations

C.4.1 Computations on Attribute Data

Operation	Function	Package	File	Line
Matrix multiplication	`%*%`		3.2.1	22
Compute *a* mod(*b*)	`%%`		6.3.3	112
Eliminate rows of a data frame not satisfying a criterion	`which()`		6.3.3	112
Iterate using a *for* loop	`for()`		2.3.1	3
Evaluate a conditional expression in a subscript	`"["`		2.3.1	21
Round a numeric value	`round()`		4.2.1	26
Generate a pseudorandom number seed	`set.seed()`		2.3.2	2
Generate a sequence of normally distributed random numbers	`rnorm()`		2.3.2	3
Generate a sequence of binomially distributed random numbers	`rbinom()`		3.2.2	6
Replicate a sequence	`rep()`		2.3.2	5
Sort a sequence of numbers	`sort()`		2.3.2	7
Test whether the elements of two arrays are all equal	`all.equal()`		3.6.2	10
Permute the elements of a sequence	`sample()`		3.4	29
Generate permutations of a sequence of numbers	`permutations()`	`e1071`	15.4.1	72

(*Continued*)

Operation	Function	Package	File	Line
Add a small random perturbation to each element of an array	`jitter()`		12.4	23
Carry out a Monte Carlo simulation by replicating an “experiment”	`replicate()`		3.3	9
Assign a value based on the outcome of a test	`ifelse()`		3.2.2	15
Apply a function to a matrix by row or by column	`apply()`		5.4	32
Apply a function to one sequence as indexed by another sequence	`tapply()`		2.3.2	10
Apply a function to a vector, creating a list	`lapply()`		4.3	8
Apply a function to a vector, creating a vector	`sapply()`		16.3.2	42
Apply a function to a data frame split by factors	`by()`		11.5.2	5
Aggregate the contents of a data frame	`aggregate()`		7.2	150
Average quantities over factors	`ave()`		9.3.4	66
Compute the distances between entries in a matrix	`vegdist()`	`vegan`	11.4.1	9
Carry out a *t*-test	`t.test()`		3.4	6
Test for independence in a contingency table using a chi-square test	`chisq.test()`		11.3.1	27
Test for independence in a contingency table using the Fisher test	`fisher.test()`		11.3.2	166
Compute a correlation coefficient	`cor()`		6.3.3	50
Test the significance of a correlation coefficient	`cor.test()`		11.2.1	10
Compute the variance	`var()`		3.3	11
Determine influence measures of a regression	`influence.measures()`		8.3	120
Plot a regression line	`abline()`		5.7	51
Carry out a bootstrap analysis	`boot()`	`boot`	10.2	50
Fit a linear regression	`lm()`		3.2.1	94
Fit a linear regression with higher order terms	`lm()`		3.2.1	125
Carry out an analysis of variance	`aov()`		16.2	79
Carry out an analysis of covariance	`aov()`		12.1	20
Test for homogeneity of variance of a linear regression	`bptest()`	`lmtest`	8.3	129
	`leveneTest()`	`car`	11.2.4	10
Computing the AIC values resulting from dropping terms from a model	`drop1()`		8.3	61

(*Continued*)

Operation	Function	Package	File	Line
Computing the AIC values resulting from adding terms to a model	`add1()`		8.4.2	24
Construct added variable plots	`avPlots()`	`car`	8.2.2	68
Construct partial residual plots	`crPlots()`	`car`	8.3	124
Compute predicted values of a regression model	`predict()`		3.2.1	96
Compute separate linear regression models over a grouped variable	`lmList()`	`nlme`	12.1	15
Carry out *k*-means cluster analysis	`kmeans()`		12.6.1	7
Carry out a recursive partitioning analysis	Several	`rpart`	9.3.1	18
Carry out a random forest analysis	Several	`randomForest`	9.4.1	5–12
Compute a logistic regression	`glm()`		8.4.1	6
Fit a zero-inflated Poisson regression model	`zeroinfl()`	pscl	8.4.4	187
Mixed model analysis	`lme()`	`nlme`	12.3	5
Generalized additive model analysis	`gam()`	`mgcv`	9.2	79

C.4.2 Computations on Spatial and Spatiotemporal Data

Operation	Function	Package	File	Line
Compute a `gstat` variogram	`variogram`	`gstat`	4.6.1	39
Fit a `gstat` variogram model	`fit.variogram()`	`gstat`	4.6.1	40
Plot a `gstat` variogram and model	`plot()`	`gstat`	6.3.2	58
Calculate an experimental correlogram	`correlogram()`	`spatial`	11.2.2	24
Compute an IDW interpolation using `gstat`	`idw()`	`gstat`	6.3.1	86
Compute a kriging interpolation using `gstat`	`krige()`	`gstat`	6.3.2	80
Compute a co-kriging interpolation using `gstat`	`Several`	`gstat`	6.3.3	105
Construct a `gstat` object	`gstat()`	`gstat`	6.3.3	104
Fit data using median polish	`medpolish()`	`spdep`	3.2.1	103

Compute and test Moran's *I*	`moran()`
	`moran.test()`
	`moran.mc()`
		`spdep`	4.4	16
				17
				26

Compute and test Geary's *c*	`geary.test()`
	`geary.mc()`
		`spdep`	4.4	28
				37

Construct a Moran correlogram	`sp.correlogram()`	`spdep`	4.5.1	16
Construct a Moran scatterplot	`moran.plot()`	`spdep`	4.5.2	14
Compute a geographically weighted regression	`gwr()`	`spgwr`	4.5.4	13

Construct a time sequence	`ISOdate()`
	`as.POSIXct()`
			15.2.1
			15.2.2
				81
				70

Construct an `stfdf` object	`STFDF()`	`spacetime`	15.2.1	89
Construct a spatiotemporal variogram	`Several`	`spacetime`	15.2.1	108