Workshop on the
R Statistical Package

Instructor:	David Lieske, Ph.D. Candidate
Time:	12:00pm to 4:30pm
Date:	Wednesday, Sept. 27
Format:	In the computer lab

Workshop Overview

Day 1

- Give a brief overview of the R Statistical Package: its strengths, weaknesses, and the type of problems it's ideal for.
- Introduce you to the software and its language (or: "why it's so powerful").
- Discuss data queries, importation, manipulation, and inspection.
- Explore R's powerful graphical tools, using distributional data for the American Crow (AMCR).

Day 2

- Dive into the details of modeling in R – how to specify models, how to retrieve output from models.
- Discuss aspects of model fit using the ANOVA and logistic regression as examples.
- Discuss aspects of model selection.
- Build a base distribution model for the AMCR.
- Build a GWR distribution model for the AMCR.

Worksheet #1

PART 1

- *R* is really a language. In this sense it's much more than a software program because once you've mastered the basic nuts & bolts you can solve any problem that comes your way – the best way to get a feel for it is to just try it out!

- Where can you get it? Visit the following site: http://www.r-project.org

- Some of *R*'s strongest features are:

1. A very compact and flexible language – that means you can do more with less lines of code (really, there isn't a problem that can't be tackled…).
2. The *R* language is object-oriented -- everything that is created in *R*, whether a variable or a matrix or what-have-you, is retrievable by name and has properties (e.g., labels).
3. Object-oriented design also applies to the output from *R* – you can grab anything you need and recombine it.
4. *R* is ideal for simulation and modeling work, because you can re-run tasks many times and dynamically interact with the results at each iteration.
5. *R* has beautiful graphical output.
6. *R* has a progressive and generous user community who are always more than ready to offer advice – many, many state-of-the-art packages are available.
7. Free of charge!

- The language is very compact and elegant, but not necessarily easy to learn or read. Different programmers have different styles, for example, the way that they name the various objects in *R*. You'll naturally develop your own style so don't worry too much about it. However, try to be consistent and systematic.

- Back to the object-oriented aspect of R…

  In addition to simple objects like arrays, *R* also uses functions, which are also objects – but ones that can execute code. Non-function objects cannot execute commands for you.

Fundamentals: Objects and Assignment

- The Assignment Operator

  ** The simplest possible example of the use of the assignment operator "<-" involves a scalar (single value) assigned to an object (e.g., "x"):

```
x <- 5
```

  The following command line creates a vector object called "x" and assigns to it the set of numbers combined together by the "c()" function.

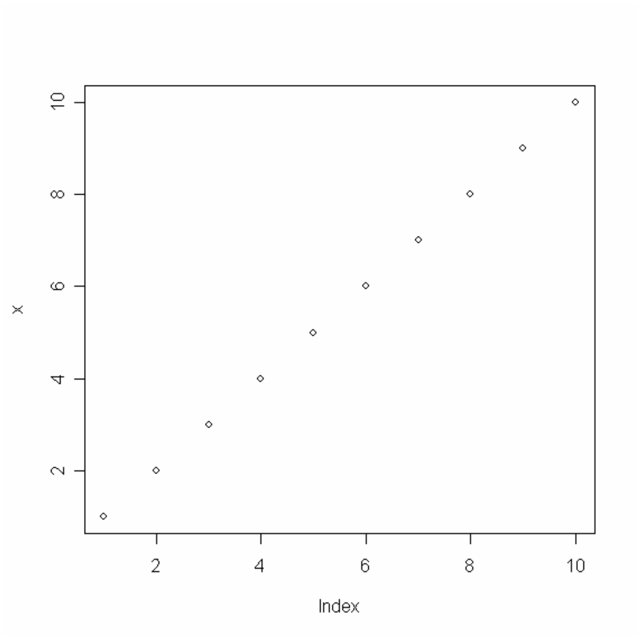
```
x <- c(1,2,3,4,5,6,7,8,9,10)
```

  It's no problem to retrieve the values of object "x" – just type it's name!

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

It’s quite easy to generate a simple scatterplot of the values of object “x” – just use the “plot()” function:

```
> plot(x)
```



Here’s another function I’d like to give you the heads-up about, because it’s really helpful for querying information about more complicated objects like lists or dataframes – it’s the “str()” function:

```
> str(x)
 num [1:10] 1 2 3 4 5 6 7 8 9 10
```

Object Queries

- Ok, what if you want to plot only the last 5 values in your vector object “x”? How would you go about querying subsets of the data?

```
> x[5:10]
[1]  5  6  7  8  9 10
```

- Now if you want to plot just this subset you can by passing it to the plot() function:

```
plot(x[5:10])
```

- Some other simple queries…

```
> x[x>8]
[1]  9 10

> x[x>8 | x<2]
[1]  1  9 10

> x[x>8 & x<2]
numeric(0)

> x[-5]
[1]  1  2  3  4  6  7  8  9 10
```

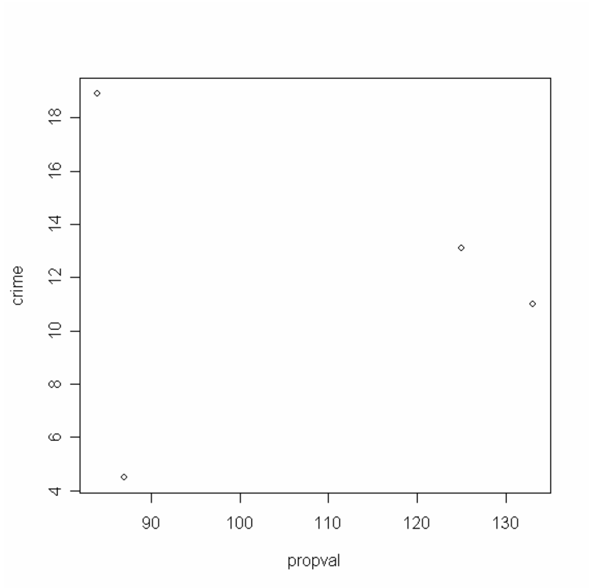
- We've now discussed two types of queries –

  (1) queries based on <u>position</u> in the vector (e.g., the 5th value, `x[5]`);

  (2) queries based on the <u>values</u> in the vector (e.g., all values in the "x" object greater than eight, `x[x>8]`).

- Ok, let's pretend that object "x" was really a variable like crime rate – let's create a variable for this called "crime", and while we're at it, let's also create a predictor (independent) variable called "propval" (to represent property value):

```
> crime <- c(13.1,4.5,11.0,18.9)
> crime
[1] 13.1  4.5 11.0 18.9

> propval <- c(125,87,133,84)
> propval
[1] 125  87 133  84

> plot(propval,crime)
```



- How did I know that I could produce a two-way scatter plot (propval vs. crime)? This would be a good time to mention *R*'s help menu…

```
> ?plot
```

```
plot                package:graphics                R Documentation

Generic X-Y Plotting

Description:

     Generic function for plotting of R objects.  For more details
     about the graphical parameter arguments, see 'par'.

Usage:

     plot(x, y, ...)

Arguments:

       x: the coordinates of points in the plot. Alternatively, a
          single plotting structure, function or _any R object with a
          'plot' method_ can be provided.

       y: the y coordinates of points in the plot, _optional_ if 'x' is
          an appropriate structure.

     ...: graphical parameters can be given as arguments to 'plot'.
```

Many methods will also accept the following arguments:

'type' what type of plot should be drawn. Possible types are

* '"p"' for *p*oints,

* '"l"' for *l*ines,

* '"b"' for *b*oth,

* '"c"' for the lines part alone of '"b"',

* '"o"' for both "*o*verplotted",

* '"h"' for "*h*istogram" like (or "high-density") vertical lines,

* '"s"' for stair *s*teps,

* '"S"' for other *s*teps, see _Details_ below,

* '"n"' for no plotting.

All other 'type's give a warning or an error; using, e.g., 'type = "punkte"' being equivalent to 'type = "p"' for S compatibility.

'main' an overall title for the plot: see 'title'.

'sub' a sub title for the plot: see 'title'.

'xlab' a title for the x axis: see 'title'.

'ylab' a title for the y axis: see 'title'.

Details:

For simple scatter plots, 'plot.default' will be used. However, there are 'plot' methods for many R objects, including 'function's, 'data.frame's, 'density' objects, etc. Use 'methods(plot)' and the documentation for these.

The two step types differ in their x-y preference: Going from (x1,y1) to (x2,y2) with x1 < x2, 'type = "s"' moves first horizontal, then vertical, whereas 'type = "S"' moves the other way around.

```
See Also:

     'plot.default', 'plot.formula' and other methods; 'points',
     'lines', 'par'.

Examples:

     plot(cars)
     lines(lowess(cars))

     plot(sin, -pi, 2*pi)

     ## Discrete Distribution Plot:
     plot(table(rpois(100,5)), type = "h", col = "red", lwd=10,
          main="rpois(100,lambda=5)")

     ## Simple quantiles/ECDF, see ecdf() {library(stats)} for a better one:
     plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type = \"s\")")
     points(x, cex = .5, col = "dark red")
```

- Ok, after a while you end up with a lot of objects floating around. How do you list them?

```
> objects()
[1] "crime"   "propval" "x"
```

- Great, let's say I'm done for the day. How do I quit *R* and shut it down gracefully?

```
> q()
```

- You keeners in the audience may be worrying about how many commands you're likely to type in a given *R* session – this is a valid concern! In addition to not wanting to have to re-type everything, you may want to review or audit the analysis later on, and for this you need a way to document what you've done.

  I use Windows "notepad" to store all of my commands, and I name all of my files in this way:

```
script.R
```

The ".R" extension tells me it's an *R* language scripting file, just like ".SPS" is an SPSS syntax file.

You can also retrieve all the commands you've typed in an *R* session using the "history()" function:

```
> history(max.show=5)

pairs(~df$DTR + df$PRECIP + df$TEMP)
history
history()
?history
history(max.show=2)
history(max.show=5)
```

PART 2

- The simplest way to import data into *R* is to use one of the standard ASCII data storage formats, namely <u>comma</u> or <u>tab-delimited</u> text files. There are at least three different functions you can use to import this type of data:

```
read.csv(file, header = TRUE, sep = ",", quote="\"",
  dec=".",fill = TRUE, ...)

read.delim(file, header = TRUE, sep = "\t",
  quote="\"", dec=".", fill = TRUE, ...)

read.table(file, header = FALSE, sep = "", quote =
  "\"'", dec = ".", row.names, col.names, as.is =
  FALSE, na.strings = "NA", colClasses = NA,
  nrows = -1, skip = 0, check.names = TRUE,
  fill = !blank.lines.skip, strip.white = FALSE,
  blank.lines.skip = TRUE, comment.char = "#",
  allowEscapes = FALSE)
```

- Let’s import the American Crow data. This data set contains occurrence data for 2799 survey points aggregated from 1997-2003 Breeding Bird Survey data for the boreal-hardwood transition zone.

- First have a look at the data in your favourite text editor to confirm how it’s been formatted:

```
xcoord     ycoord     presence  …
1621035.14     4973929.11     1     …
1620893.71     4974728.34     1     …
```

- Now let’s use the `read.table()` function to import the data now that we know that it’s a tab-delimited text file:

```
> read.table("amcr.txt", sep="\t")
```

  What did you see?

  Let’s revise the previous command execution to assign the results of “read.table()” to an object called “df”:

```
> df <- read.table("amcr.txt", sep="\t", header=TRUE)
```

```
> str(df)
```

```
> str(df)
`data.frame':   2799 obs. of  13 variables:
 $ xcoord  : num  1621035 1620894 1620757 1620629 1616523 ...
 $ ycoord  : num  4973929 4974728 4975528 4976330 4976492 ...
 $ CROPVEG : int  1 1 0 0 0 0 0 0 0 0 ...
 $ CONIFER : int  0 0 0 0 0 0 0 0 0 0 ...
 $ DECID   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ MIXEDF  : int  0 0 1 1 0 1 1 1 1 1 ...
 $ EVIMEAN : num  5226 5226 4800 4270 5189 ...
 $ EVISD   : num  188 188 502 718 424 ...
 $ ELEV    : int  328 322 322 323 336 335 328 326 322 320 ...
 $ DTR     : num  11.1 11.1 11.1 11.1 11.1 ...
 $ PRECIP  : num  730 730 730 730 730 ...
 $ TEMP    : num  7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 ...
 $ presence: int  1 1 1 0 1 0 1 1 1 1 ...
```

- What are these variables? That’s where metadata comes in handy…

Table 1. Candidate variables used to model species distributions.

Variable	Description	Source
Land Cover		
CONIFER	Conifer-dominated forest	MODIS 2001 land cover classification [*]
CROPVEG	Cropland/natural vegetation mosaic	MODIS 2001 land cover classification[*]
DECID	Deciduous-dominated forest	MODIS 2001 land cover classification [*]
MIXEDF	Mixed (conifer-deciduous) forest	MODIS 2001 land cover classification [*]
EVIMEAN	Mean Enhanced Vegetation Index (EVI) for a 3x3 neighbourhood	MODIS 16-day EVI [**]
EVISD	Mean Standard Deviation of the Enhanced Vegetation Index (EVI) for a 3x3 neighbourhood	MODIS 16-day EVI [**]
Climate		
DTR	Mean diurnal temperature range (^{0}C)	CRU_2_10 global climate data [***]
PRECIP	Total annual precipitation (mm)	CRU_2_10 global climate data [***]
TEMP	Mean temperature (^{0}C)	CRU_2_10 global climate data [***]
Topography		
ELEV model[****]	Elevation (m)	GTOPO30 global digital elevation

[*]Derived from the MOD12Q1 (1-km resolution) layer incorporating the 17-class land cover classification system defined by the International Geosphere-Biosphere Programme (IGBP), U.S. Geological Survey (2006).

[**]Derived from the MOD13A2 (1-km resolution) layer for 25 June, 2004, U.S. Geological Survey & NASA (2006).

[***]Monthly averages (or annual totals, in the case of precipitation) were averaged over the years 1997-2002 at a resolution of 0.5^0 x 0.5^0 (Mitchell & Jones 2005).

[****]Global digital elevation data at an approximately 1-km resolution (U.S. Geological Survey 1996).

- Ok, what kind of object is “df”? It’s a data frame which is *R*’s standard format for representing data – Note that data frame objects are just like Excel spreadsheets. The have variables (with names), and data points are represented as rows in the table. You can manually edit data frames (if you need to, but *R* isn’t the best place to do this…):

```
> edit(df)
```

- Let's explore the variables in this dataset, starting with the response variable:

  `presence`

  It's a binary variable (1/0) indicating presence at anytime during the 1997-2003 period. Because it's a categorical (dummy) variable with a small number of classes, it's best summarized using a simple cross-tabulation:

  ```
  > table(df$presence)

     0    1
  1581 1218
  ```

  `table()` tabulated 1218 presence points (presence = 1) out of 2799 survey points (43.5% of locations).

  We'll explore further functions for summarizing and visualizing this data in Part 3.

PART 3

- What proportion of the survey points fall into the different land cover classes?

  ```
  > table(df$CROPVEG)

     0    1
  2355  444

  > table(df$CONIFER)

     0    1
  2600  199

  > table(df$DECID)

     0    1
  2798    1

  > table(df$MIXEDF)

     0    1
  1242 1557
  ```

Variable	**% of Survey Points**
CROPVEG	444/2799 = **15.9%**
CONIFER	199/2799 = **7.1%**
DECID	1/2799 = **0.04%**
MIXEDF	1557/2799 = **55.6%**

- What proportion of the AMCR presence points occur in the previous land cover types, and is their evidence of selection for or against each of the classes?

CROPVEG

```
> table(df$presence,df$CROPVEG);
chisq.test(df$presence,df$CROPVEG)

       0    1
  0 1415  166
  1  940  278

        Pearson's Chi-squared test with Yates' continuity
correction

data:  df$presence and df$CROPVEG
X-squared = 77.3781, df = 1, p-value < 2.2e-16
```

CONIFER

```
> table(df$presence,df$CONIFER);
chisq.test(df$presence,df$CONIFER)

       0    1
  0 1431  150
  1 1169   49

        Pearson's Chi-squared test with Yates' continuity
correction
```

```
data:  df$presence and df$CONIFER
X-squared = 30.287, df = 1, p-value = 3.726e-08
```

DECID

```
> table(df$presence,df$DECID); chisq.test(df$presence,df$DECID)

       0    1
  0 1580    1
  1 1218    0

        Pearson's Chi-squared test with Yates' continuity
correction

data:  df$presence and df$DECID
X-squared = 0.0171, df = 1, p-value = 0.896

Warning message:
Chi-squared approximation may be incorrect in:
chisq.test(df$presence, df$DECID)
>
```

MIXEDF

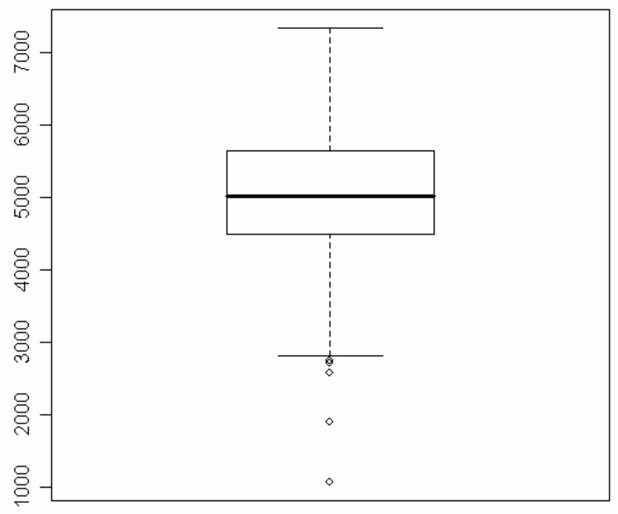
```
> table(df$presence,df$MIXEDF); chisq.test(df$presence,df$MIXEDF)

      0   1
  0 642 939
  1 600 618

        Pearson's Chi-squared test with Yates' continuity
correction

data:  df$presence and df$MIXEDF
X-squared = 20.5243, df = 1, p-value = 5.888e-06
>
```

Variable	**% of Survey Points**	**% of AMCR Presence Points**
CROPVEG	444/2799 = **15.9%**	278/1218 = **22.8% (+)**
CONIFER	199/2799 = **7.1%**	49/1218 = **4.0% (-)**
DECID	1/2799 = **0.04%**	0/1218 = **0% (-)**
MIXEDF	1557/2799 = **55.6%**	600/1218 = **49.3% (-)**

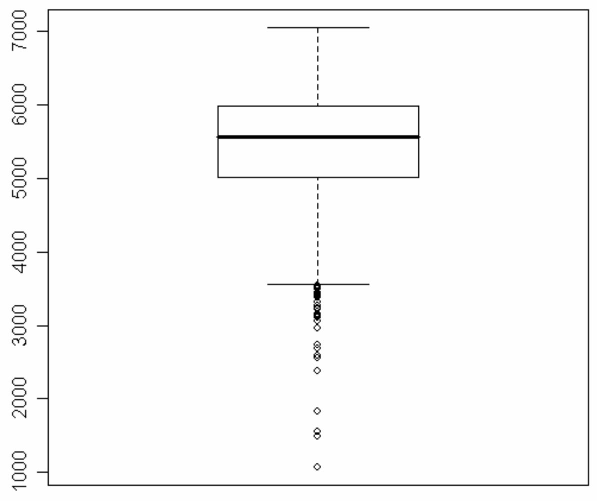
- What is the relationship between occurrence of AMCR and the continuous variables `EVIMEAN, EVISD, DTR, PRECIP, TEMP, ELEV?`

Let's start with the `EVIMEAN` variable, but let's look at its relationship with absence and presence separately:

```
> summary(df$EVIMEAN[df$presence == 0])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1069    4495    5018    5078    5644    7343
> boxplot(df$EVIMEAN[df$presence == 0])
```



```
> summary(df$EVIMEAN[df$presence == 1])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1069    5015    5558    5391    5989    7052
> boxplot(df$EVIMEAN[df$presence == 1])
```



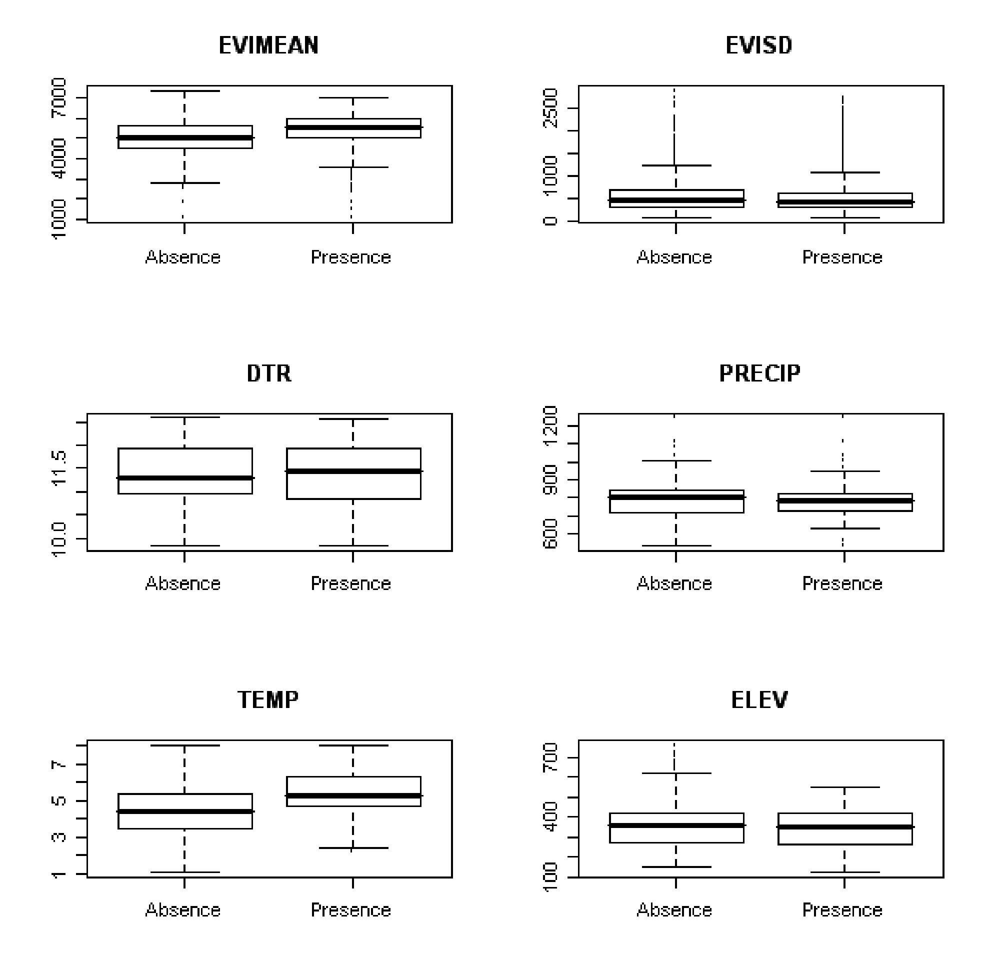
There is a way to change the default graphing parameters in *R* to customize the formatting of the output. In this case, I want to simultaneously view the previous output, for all six variables. I'll use a 3x2 panel to achieve this:

```
> par.old <- par()
> par(mfrow=c(3,2))
> label <- c("Absence", "Presence")
> boxplot(df$EVIMEAN[df$presence==0],
df$EVIMEAN[df$presence == 1],main="EVIMEAN")
> axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)
```

... [just copy and paste the following block of commands]

```
boxplot(df$EVISD[df$presence==0], df$EVISD[df$presence
== 1],main="EVISD")
axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)

boxplot(df$DTR[df$presence==0], df$DTR[df$presence ==
1],main="DTR")
axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)
```

```
boxplot(df$PRECIP[df$presence==0],
df$PRECIP[df$presence == 1],main="PRECIP")
axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)

boxplot(df$TEMP[df$presence==0], df$TEMP[df$presence
== 1],main="TEMP")
axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)

boxplot(df$ELEV[df$presence==0], df$ELEV[df$presence
== 1],main="ELEV")
axis(side=1, at=c(1,2), labels=label, lwd=1, font=1)
```

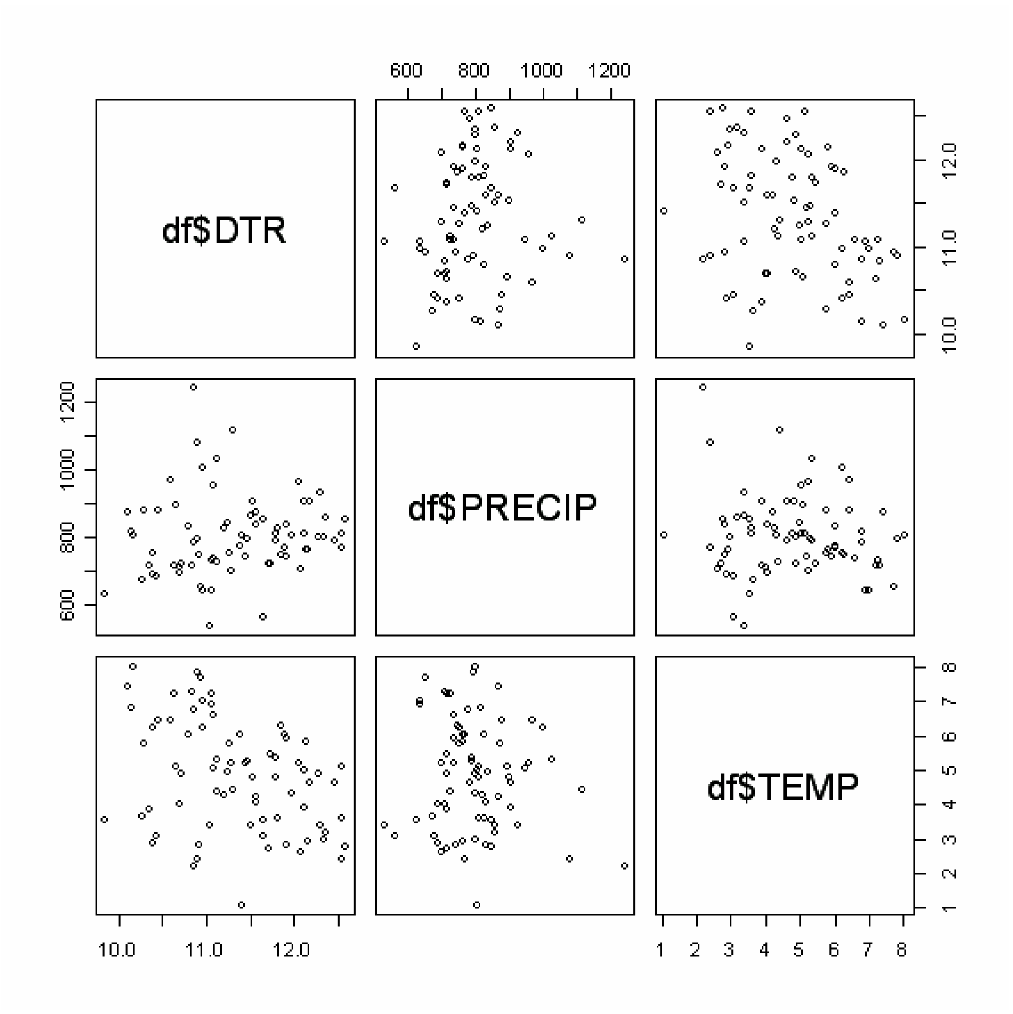


- Now that we've finished producing our 3x2 panel, you can reset the original graphing parameters this way:

```
> par(par.old)
```

- Another useful graphical exploratory data analysis (EDA) tool is the pairwise plot applied to continuous variables:

```
> pairs(~df$DTR + df$PRECIP + df$TEMP)
```



APPENDIX: Some notes for further exploration

- Exporting graphical output from *R* is pretty easy – if you right click on the image you can choose to either copy it, or save it. If you intend to copy and paste the image as I did in this worksheet, use the “copy to bitmap” option – it produces quite nice imbedded graphics. Another option is to save the image as a "encapsulated postscript", which you can open into, for example, Adobe Illustrator. You can resize particular elements to your hearts content, as well as modify bits of the image. You could then export the finished product back to a bitmap, or some other format (GIFF, JPEG, etc.).

- You can export data from dataframes in a fashion analogous to importing data, using the "write.table()" function. For example, let's say I have a dataframe called "data" that I want to export to a comma-delimited file called "export.txt" (with the first row containing the variable names):

  ```
  > write.table(data, file="export.txt", sep=",", col.names=TRUE)
  ```

Workshop on the
<u>*R* Statistical Package</u>

Instructor:	David Lieske, Ph.D. Candidate
Time:	9:00am to 12:00pm
Date:	Thursday, Sept. 28
Format:	In the computer lab

Workshop Overview

Day 1

- Give a brief overview of the R Statistical Package: its strengths, weaknesses, and the type of problems it's ideal for.
- Introduce you to the software and its language (or: "why it's so powerful").
- Discuss data queries, importation, manipulation, and inspection.
- Explore R's powerful graphical tools, using distributional data for the American Crow (AMCR).

Day 2

- Dive into the details of modeling in R – how to specify models, how to retrieve output from models.
- Discuss aspects of model fit, using the ANOVA and logistic regression as examples.
- Discuss aspects of model selection.
- Build a base logistic distribution model for the AMCR.
- Build a GWR distribution model for the AMCR.

Worksheet #2

PART 1 – Model Specification in *R*

- *R* has its own symbolic language for representing statistical models.

- For a simple linear model, such as a linear regression with one intercept and one slope parameter, the formula would be represented as:

```
y ~ b
```

Note that you don’t need to specify the intercept because it’s assumed by default. If, for some reason, you do not wish to estimate the intercept, you would write the following:

```
y ~ b – 1
```

When specifying a multivariate regression, you combine each parameter to be estimated in a simple additive fashion:

```
y ~ b1 + b2 + b3
```

To specify a polynomial regression you have a couple of options at your disposal: (1) you can use the “poly()” function, or (2) you can create the polynomials manually using the “I()” function:

```
y ~ poly(b,2)
```

means the same thing as:

```
y ~ b + I(b^2)
```

Basically, the “I()” function allows you to perform arithmetic operations on the variable specified within the parenthesis – in this case the square of variable `b`.

** Another thing you need to know about: what if your predictor variable is a factor with discrete levels (e.g., drug A, B or C), and not a continuous measurement?

```
yield <- c(60,72,54,68,52,83,45,80)
```

```
cat <- c("A","A","A","A","B","B","B","B")

cat <- as.factor(cat)
cat
[1] A A A A B B B B
Levels: A B
```

Now you would proceed with specifying the model the same way you did with the continuous measurements:

```
yield ~ cat
```

What if you have reason to believe that the effect of one variable is moderated or influenced by the presence of another variable, in other words, an **interaction**. How would you model this in *R*?

```
y ~ A + B + A:B
```

or more compactly as:

```
y ~ A*B
```

So let's add a third variable called "temp" and illustrate how we could specify a model that incorporates the interaction between "temp" and "cat":

```
temp <- c(“160”,”180”,”160”,”180”,”160”,”180”,
  ”160”,”180”)

temp <- as.factor(temp)

yield ~ cat + temp + cat:temp
```

or:

```
yield ~ cat*temp
```

- Let’s use the “yield” data from the previous example to illustrate how to estimate model parameters in *R*,in this case using a classical Analysis of Variance (ANOVA) approach.

  But first, let’s do a little housekeeping with the “yield”, “cat”, and “temp” variables. Let’s join them together into one single object:

```
df <- cbind(yield,cat,temp)

> df
     yield cat temp
[1,]    60   1    1
[2,]    72   1    2
[3,]    54   1    1
[4,]    68   1    2
[5,]    52   2    1
[6,]    83   2    2
[7,]    45   2    1
[8,]    80   2    2


> df <- as.data.frame(df)
> df
  yield cat temp
1    60   1    1
2    72   1    2
3    54   1    1
4    68   1    2
5    52   2    1
6    83   2    2
7    45   2    1
8    80   2    2


> df$cat <- factor(cat,levels=c("A","B"))
> df$temp <- factor(temp,levels=c("160","180"))
> df
  yield cat temp
1    60   A  160
2    72   A  180
3    54   A  160
4    68   A  180
5    52   B  160
6    83   B  180
```

```
7    45   B  160
8    80   B  180
```

Great! This may not have looked like a simple procedure, but it makes for a much tidier workspace – which means less chance of you (as the analyst) making a silly mistake.

To perform our Analysis of Variance we'll use the function "aov()". It's basic arguments are:

```
aov(formula, data)
```

In our case, we want to call "aov()" in this way (<u>making sure to store the results in an object called "cat.aov"</u>)

```
cat.aov <- aov(yield ~ cat*temp, data=df)
```

```
> cat.aov
Call:
   aov(formula = yield ~ cat * temp, data = df)

Terms:
                  cat   temp cat:temp Residuals
Sum of Squares     4.5 1058.0    200.0      55.0
Deg. of Freedom      1      1        1         4

Residual standard error: 3.708099
Estimated effects may be unbalanced
```

```
> summary(cat.aov)
            Df  Sum Sq Mean Sq F value    Pr(>F)    
cat          1    4.50    4.50  0.3273 0.5978852    
temp         1 1058.00 1058.00 76.9455 0.0009312 ***
cat:temp     1  200.00  200.00 14.5455 0.0188776 *  
Residuals    4   55.00   13.75                      
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 
0.1 ' ' 1
```

Let’s have a look at the raw data:

```
boxplot(df$yield[df$cat == "A"],
  df$yield[df$cat == "B"], names=c("A","B"),
  main=”Yield by Type of Catalyst”)


boxplot(df$yield[df$temp == "160"],
  df$yield[df$temp == "180"], names=c("160","180"),
  main=”Yield by Temperature”)
```

PART 2 – Specification of the Base Logistic Regression for the AMCR

- Let’s return to the AMCR data from worksheet #1 (day 1):

```
df <- read.table("amcr.txt", sep="\t", header=TRUE)
```

Let’s double check the data frame to make sure everything’s where it’s supposed to be:

```
> dim(df)
[1] 2799   13

> str(df)
`data.frame':   2799 obs. of  13 variables:
 $ xcoord  : num  1621035 1620894 1620757 1620629 1616523 ...
 $ ycoord  : num  4973929 4974728 4975528 4976330 4976492 ...
 $ CROPVEG : int  1 1 0 0 0 0 0 0 0 0 ...
 $ CONIFER : int  0 0 0 0 0 0 0 0 0 0 ...
 $ DECID   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ MIXEDF  : int  0 0 1 1 0 1 1 1 1 1 ...
 $ EVIMEAN : num  5226 5226 4800 4270 5189 ...
 $ EVISD   : num  188 188 502 718 424 ...
 $ ELEV    : int  328 322 322 323 336 335 328 326 322 320 ...
 $ DTR     : num  11.1 11.1 11.1 11.1 11.1 ...
 $ PRECIP  : num  730 730 730 730 730 ...
 $ TEMP    : num  7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 ...
 $ presence: int  1 1 1 0 1 0 1 1 1 1 ...
```

We’re interested in modeling the effect of land cover (“CROPVEG”, “CONIFER”, “DECID”, “MIXEDF”, “EVIMEAN”, “EVISD”), climactic (“DTR”, “PRECIP”, and “TEMP”), and topographic (“ELEV”) variables on the probability of occurrence for the AMCR.

```
presence ~ CROPVEG + CONIFER + DECID + MIXEDF +
           EVIMEAN + EVISD + DTR + PRECIP + TEMP +
           ELEV
```

The *R* function for calculating logistic regressions (“glm()”) is part of the Generalized Linear Models family with a logistic-link. Let’s determine what arguments it takes:

```
?glm

Fitting Generalized Linear Models

Description:

     'glm' is used to fit generalized linear models, specified by
     giving a symbolic description of the linear predictor and a
     description of the error distribution.

Usage:

     glm(formula, family = gaussian, data, weights, subset,
         na.action, start = NULL, etastart, mustart,
         offset, control = glm.control(...), model = TRUE,
         method = "glm.fit", x = FALSE, y = TRUE, contrasts =
NULL, ...)
```

As with the Analysis of Variance example above, when we call the function “glm()” we want to assign the result to an object so that we can view it and manipulate it.

```
df.glm <- glm(formula=presence ~ CROPVEG + CONIFER +
  DECID + MIXEDF + EVIMEAN + EVISD + DTR + PRECIP +
  TEMP + ELEV, family=binomial, data=df)

> df.glm

Call:  glm(formula = presence ~ CROPVEG + CONIFER + DECID + MIXEDF +      EVIMEAN
+ EVISD + DTR + PRECIP + TEMP + ELEV, family = binomial,      data = df)

Coefficients:
(Intercept)      CROPVEG      CONIFER        DECID       MIXEDF      EVIMEAN
 -8.321e+00    1.763e-01    1.572e-01   -1.045e+01   -1.700e-01    2.377e-04
      EVISD          DTR       PRECIP         TEMP         ELEV
  3.586e-04    3.007e-01    1.384e-04    4.851e-01    1.946e-03

Degrees of Freedom: 2798 Total (i.e. Null);  2788 Residual
Null Deviance:      3833
Residual Deviance: 3472        AIC: 3494
```

```
> str(df.glm)
```

```
List of 30
 $ coefficients     : Named num [1:11]  -8.321   0.176   0.157 -10.446  -0.170 ...
  ..- attr(*, "names")= chr [1:11] "(Intercept)" "CROPVEG" "CONIFER" "DECID" ...
 $ residuals        : Named num [1:2799]  1.47  1.48  1.67 -2.42  1.52 ...
  ..- attr(*, "names")= chr [1:2799] "1" "2" "3" "4" ...
 $ fitted.values    : Named num [1:2799] 0.678 0.676 0.598 0.587 0.659 ...
  ..- attr(*, "names")= chr [1:2799] "1" "2" "3" "4" ...
 $ effects          : Named num [1:2799]  5.2082 -7.8648 -4.3641 -0.0572 -2.1948
...
  ..- attr(*, "names")= chr [1:2799] "(Intercept)" "CROPVEG" "CONIFER" "DECID" ...
 $ R                : num [1:11, 1:11] -24.6   0.0   0.0   0.0   0.0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:11] "(Intercept)" "CROPVEG" "CONIFER" "DECID" ...
  .. ..$ : chr [1:11] "(Intercept)" "CROPVEG" "CONIFER" "DECID" ...
 $ rank             : int 11
 $ qr               :List of 5
  ..$ qr   : num [1:2799, 1:11] -24.5670   0.0191   0.0200   0.0200   0.0193 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2799] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:11] "(Intercept)" "CROPVEG" "CONIFER" "DECID" ...
  ..$ rank : int 11
  ..$ qraux: num [1:11] 1.02 1.04 1.01 1.00 1.03 ...
  ..$ pivot: int [1:11] 1 2 3 4 5 6 7 8 9 10 ...
  ..$ tol  : num 1e-11
  ..- attr(*, "class")= chr "qr"
 $ family           :List of 11
  ..$ family    : chr "binomial"
  ..$ link      : chr "logit"
  ..$ linkfun   :function (mu)
  ..$ linkinv   :function (eta)
  ..$ variance  :function (mu)
  ..$ dev.resids:function (y, mu, wt)
  ..$ aic       :function (y, n, mu, wt, dev)
  ..$ mu.eta    :function (eta)

….

 $ deviance         : num 3472
 $ aic              : num 3494
 $ null.deviance    : num 3833
 $ iter             : int 10
 $ weights          : Named num [1:2799] 0.218 0.219 0.240 0.242 0.225 ...
  ..- attr(*, "names")= chr [1:2799] "1" "2" "3" "4" ...
 $ prior.weights    : Named num [1:2799] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "names")= chr [1:2799] "1" "2" "3" "4" ...
 $ df.residual      : int 2788
 $ df.null          : int 2798
```

……

Ok, so we know the model was successfully estimated, i.e., it produced coefficients for each of the predictor variables – but are they all significant? We should produce a summary of the “df.glm” object:

```
> summary(df.glm)

Call:
glm(formula = presence ~ CROPVEG + CONIFER + DECID + MIXEDF +
    EVIMEAN + EVISD + DTR + PRECIP + TEMP + ELEV, family = binomial,
    data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-1.6988  -0.9611  -0.6490   1.0952   2.0552

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -8.321e+00  9.496e-01  -8.763  < 2e-16 ***
CROPVEG      1.763e-01  1.438e-01   1.226 0.220190
CONIFER      1.572e-01  2.117e-01   0.743 0.457566
DECID       -1.045e+01  1.970e+02  -0.053 0.957705
MIXEDF      -1.700e-01  1.060e-01  -1.604 0.108722
EVIMEAN      2.377e-04  6.157e-05   3.861 0.000113 ***
EVISD        3.586e-04  1.169e-04   3.066 0.002167 **
DTR          3.007e-01  7.376e-02   4.077 4.56e-05 ***
PRECIP       1.384e-04  4.059e-04   0.341 0.733181
TEMP         4.851e-01  3.841e-02  12.629  < 2e-16 ***
ELEV         1.946e-03  4.856e-04   4.008 6.13e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3833.0  on 2798  degrees of freedom
Residual deviance: 3471.8  on 2788  degrees of freedom
AIC: 3493.8

Number of Fisher Scoring iterations: 10
```

Interesting results. Of the land cover variables, EVIMEAN and EVISD appear to be positive predictors for AMCR occurrence. Of the climactic variables, DTR and TEMP appear to also be positive predictors. ELEV appears as a positive predictor as well.

But how can we assess the goodness of fit for the linear model? One measure of fit is predictive accuracy, which we can assess using the area under the curve (AUC) of a receiver operating characteristic curve (ROC). We'll load a library off the WWW that contains a nice function for calculating AUC :

```
> library(Epi)
> ROC(test=df.glm$fitted.values, stat=df$presence)$AUC
```

<u>We find that the accuracy is: 0.703</u>.

There's another measure of fit that can be applied to logistic (binary) measurements, based on the Hosmer-Lemeshow test (see APPENDIX):

```
df.glm.hl <- hltest(resp=df$presence, prob=df.glm$fitted.values, g=5)
> df.glm.hl
```

```
$table
   group
      1   2   3   4   5
  0 424 455 271 227 204
  1 134 106 290 334 354

$Chat
[1] 65.56132

$df
[1] 3

$sig
[1] 3.808065e-14
```

Model Selection using AIC

In the previous analysis with forced all of the variables to be added to the logistic regression. Another choice would be to use an automated model selection algorithm such as “step()” which uses forward or backward stepwise addition of variables and a stopping rule based on Akaike Information Criterion (AIC) values.

What is the AIC? AIC = -2*ln(likelihood) + 2*p*

Let’s repeat the previous model estimation procedure, but this time introduce the function “step()”:

```
df.glm.step <- step(df.glm)

Start:  AIC= 3493.81
 presence ~ CROPVEG + CONIFER + DECID + MIXEDF + EVIMEAN + EVISD
+
    DTR + PRECIP + TEMP + ELEV

          Df Deviance    AIC
- PRECIP   1   3471.9 3491.9
- CONIFER  1   3472.4 3492.4
- DECID    1   3472.4 3492.4
- CROPVEG  1   3473.3 3493.3
<none>         3471.8 3493.8
- MIXEDF   1   3474.4 3494.4
- EVISD    1   3481.2 3501.2
- EVIMEAN  1   3486.9 3506.9
- ELEV     1   3487.9 3507.9
- DTR      1   3488.8 3508.8
- TEMP     1   3648.3 3668.3
```

…

```
> summary(df.glm.step)

Call:
glm(formula = presence ~ MIXEDF + EVIMEAN + EVISD + DTR + TEMP +
    ELEV, family = binomial, data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6714  -0.9649  -0.6479   1.0953   2.0816

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -8.123e+00  8.536e-01  -9.517  < 2e-16 ***
MIXEDF      -2.517e-01  8.434e-02  -2.985  0.00284 **
EVIMEAN      2.319e-04  5.908e-05   3.926 8.65e-05 ***
EVISD        3.288e-04  1.124e-04   2.925  0.00344 **
DTR          3.027e-01  7.302e-02   4.145 3.40e-05 ***
TEMP         4.898e-01  3.494e-02  14.016  < 2e-16 ***
ELEV         1.939e-03  4.829e-04   4.015 5.93e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3833.0  on 2798  degrees of freedom
Residual deviance: 3474.2  on 2792  degrees of freedom
AIC: 3488.2

Number of Fisher Scoring iterations: 4
```

Displaying Predicted Probabilities of Occurrence

- Let's load a library that will help us produce some maps:

```
library(geoR)
```

- We need to "attach" the predicted probabilities of occurrence (from the object `df.glm.step` above) to the "df" dataframe that holds our American Crow data:

```
prob <- df.glm.step$fitted.values
df <- cbind(df,prob)
```

- Now we need to produce an object that the "geoR" package can utilise:

```
df.geodata <- as.geodata(df,coords.col=1:2, data.col=14)
plot.geodata(df.geodata)
```

PART 3 – Exploration of a GWR Logistic Regression for the AMCR

- Until now we've assumed that the model relationships (e.g., a positive slope of `0.4898` for the relationship between AMCR occurrence and average temperature) is the same everywhere in the study area – is this necessarily true?

- This leads us to the <u>STATIONARITY ASSUMPTION</u>: A spatial process is stationary if its statistical properties (mean, variance) are independent of absolute location in the study region. It also implies that the relationship between values at any two points depends only on the distance and direction between them, and not on their absolute location… [but GWR doesn't really deal with the 2nd part of the assumption so we won't talk about it anymore, :>] (see Bailey and Gatrell 1995: 33).

- According to Fotheringham, Brunsdon, and Charlton (2002: 9-10), there are three reasons why relationships might vary with location:

1. Sampling variation & noise.
2. The process generating the observations is not the same everywhere.
3. Unmeasured but influential factors (or processes) are active at different locations.

In the case of the AMCR, it's pretty easy to imagine the species exhibiting differential adaptation to environmental conditions (case #2). I can also imagine #3 applying because there are bound to be environmental or anthropogenic factors active in different portions of the large study area that we don't know about. Case #1 must apply, as multiple observers conduct the survey and all observations are made with a certain amount of error.

- How does “geographically-weighted regression” work?

    1. Estimates model relationships for each individual location (=local estimates) – e.g., a data set with 350 points will have 350 estimates for each parameter!
    2. Applies a moving kernel (=moving window) which differentially weights the influence of neighbouring points based on distance to the location in question:

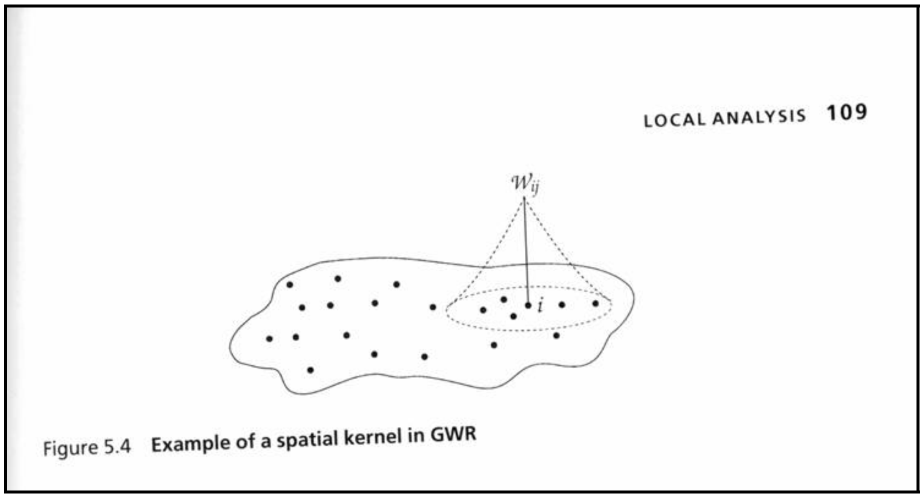


Figure 5.4 **Example of a spatial kernel in GWR**

(from: Fotheringham, Brunsdon & Charlton, 2000: 111)

- So let’s dive in and have a look at the results!
- First, we need to load the GWR library (assuming that the required script and .dll is already stored in the working directory):

```
source(“gwr4_1c.R”)
```

- The GWR package of Fotheringham, Brunsdon and Charlton, like most of the packages available for *R*, have nuances that you need to learn about before you use them. In the case of the GWR functions, you need to create matrices and pass them as arguments:

```
xvar <- c(“MIXEDF”,”EVIMEAN”,”EVISD”,”DTR”,”TEMP”,”ELEV”)

x <- as.matrix(df[,xvar])
y <- df[,"presence"]
loc <- as.matrix(df[,c("xcoord", "ycoord")])
```

- Now let’s call the “gwr.binomial()” function:

```
> df.gwr <- gwr.binomial(x,y,loc,out.loc=loc,grid=FALSE,adaptive=NULL)
Iteration    Log-Likelihood
=========================
      0        -1381
      1        -1324
      2        -1311
      3        -1310
      4        -1310
      5        -1310
      6        -1310
```

- Let’s have a look at the “df.gwr” object:

```
> df.gwr
Binomial GWR model with variables : Intercept MIXEDF EVIMEAN
EVISD DTR TEMP ELEV
Bandwidth      =  501662
Effective D.F. =  31.37
Corrected AIC  =  2683
```

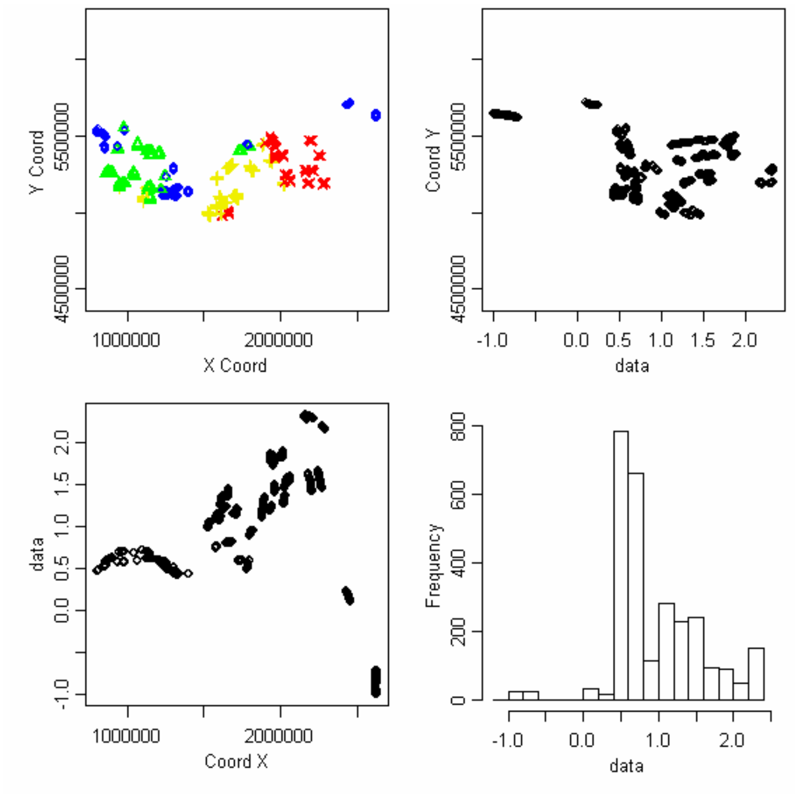
- Using TEMP as an example, let’s take a look at the range of estimates for this variable:

```
> summary(df.gwr$est[,c("TEMP")])
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.0020  0.5796  0.7035  0.9567  1.3070  2.3300
```

Remember the original estimate for TEMP? `0.4898`

- Let’s look at the map of the variation in parameter estimates:

```
> temp.est <- df.gwr$est[,c(“TEMP”)]
> df <- cbind(df,temp.est)
> df.geodata <- as.geodata(df,coords.col=1:2, data.col=15)
> plot.geodata(df.geodata)
```



- What about the accuracy of the predictions from this new, geographically-explicit model?

```
x <- as.matrix(cbind(rep(1,dim(loc)[1]), x))
b <- df.gwr$est

# Function specification
fitted <- function(x,b) apply(x*b,1,sum)

gwr.fitted <- fitted(x,b)

#N.B. You need to export "gwr.pred" to the global envt so that the
#"ROC" can find the vector...
gwr.pred <- 1 / (1 + exp(-1*gwr.fitted))


#
# Test the accuracy
#

auc <- ROC(test=gwr.pred, stat=y)$AUC

> auc
[1] 0.8561962


# the following is a goodness-of-fit test based on function "hltest()"
hllist <- hltest(resp=df$presence,prob=gwr.pred,g=5)
```

```
> hllist
$table
   group
      1   2   3   4   5
  0 548 477 272 180 104
  1  10  84 289 381 454

$Chat
[1] 67.97499

$df
[1] 3

$sig
[1] 1.154632e-14
```

APPENDIX A

- Here's the code for the function to determine the Hosmer & Lemeshow goodness-of-fit (just copy and paste it into the *R* command window, and call it using "hltest()" and three arguments):

```
hltest <- function(resp, prob, g) {

  # Function for computing C-hat = approx. distributed as a chi-square
  # with g-2 degrees of freedom (Hosmer & Lemeshow p.148)
  # This is an omnibus goodness-of-fit measure for binary (as compared
  # to grouped, binomial data) models.

  # Arguments:
  # resp = "response" = a vector of dummy variables (coded 1/0)
  #        indicating the presence of the effect
  # prob = a vector of expected probabilities for each observation, based
  #        on the model
  # g    = scalar indicating the number of groups to use (e.g. 4, 10)


  # extract the binary values and expected probabilities for the
  # data set

  hl.mat <- cbind(resp,prob)


  # sort the matrix from lowest to highest expected probabilities

  hl.mat <- hl.mat[order(hl.mat[,2]),]
  key <- seq(1:length(prob))
  hl.mat <- cbind(hl.mat, key)


  # cut the predicted probabilities into “g” equal-sized groups
  group <- factor(cut(hl.mat[,3],g))

  # label the cutoff categories as "1" through "g"
  levels(group) <- c(1:g)

  #
  # Now attach the groupings to the "hl.mat" matrix
  #

  hl.mat <- cbind(hl.mat, group)


  #
  # Hosmer & Lemeshow's Goodness-of-fit (p.148)
  #
  # n = number of observations in the quantile
  # pi-bar = average of the expected probabilities for that quantile
  # o = number of observed positive (=1) outcomes
  # y = number of "positive" responses
  #

  # Count up the number of “resp” == 1, by group
  y <- table(hl.mat[,1], hl.mat[,4])[2,]

  # Note usage of “tapply”: tapply(data,categories,function)
  n <- tapply(hl.mat[,1],hl.mat[,4],length)

  # “pi.bar” = average probability of a “resp”=1, by group
  pi.bar <- tapply(hl.mat[,2], hl.mat[,4],mean)

  c <- (y-n*pi.bar)^2/((n*pi.bar)*(1-pi.bar))


  # Return the results as a list...
  hl.list <- NULL
  hl.list <- list(table=table(hl.mat[,1], group), Chat = sum(c), df=g-2,
    sig = (1-pchisq(sum(c),g-2)))

}
```

- Oftentimes we might want to take a random subset of our data points, for example, to perform model testing. Here’s some simple code -- applied to the “df” dataframe from the AMCR – that will enable you to randomly sample 100 records.

```
# Step 1. Create a vector of rowid numbers the same size as your
# data set to be sampled.
> size <- dim(df)[1]
> rowid <- seq(1,size,by=1)

# Step 2. Sample the rowid numbers without replacement; store the
# result in “row.sample”
> row.sample <- sample(rowid,size=100,replace=FALSE)

# Step 3. Grab the subset of your original dataframe and call it
# “df.sample”
> df.sample <- df[row.sample,]

# Step 4. Double check to confirm it worked…
> dim(df.sample)
[1] 100  15
```

- Looping in *R* and an “if else” control structure:

```
for (i in 1:100) {

  cat(paste(“Entering “, i, “th loop”, sep=””),“\n”)

  if (i > 95 & i < 100) {

    cat(“*** (Almost done)”,”\n”)

  }else if (i > 99) {

    cat(“!! Done !!”,”\n”)

  }

}
```

```
#
# Name:             "allcombo.R"
# Date:             Aug.10, 2006
# Author:    David Lieske
# Purpose:   This script calls the leaps library to build an
#            "all combinations" matrix of fields for GLM model
#            building.
# Arguments: Assumes a pre-existing dataframe called "df" exists
#               in the global environment.
# Value:     A matrix called "allcombo.out", sorted by AIC values,
#            with three fields: (1) aic; (2) auc; (3) predictor
#            variable combination
#

library(leaps)
library(Epi)


#
#
# First: define the x variables dataframe
#
#

TEMPELEV <- df$TEMP*df$ELEV
df <- cbind(df,TEMPELEV)


# You need to modify this!
x.list <- c("CONIFER","CROPVEG","DECID","MIXEDF","EVIMEAN","EVISD",
  "DTR","PRECIP","TEMP","ELEV","TEMPELEV")

# “x” is a specific set of predictor variables
x <- df[,x.list]

# “x.combo” is the combination of models, assuming y = predictor variable
#(which you need to specify)
 x.combo <- leaps(x=x,y=binary)$which
```

```
#
#
# Second: use the matrix of selected variables to select a subset of
# x variables for each iteration
#
#


# The following creates a dummy data frame to hold the AIC results,
# along with their formulae

aic <- rep(0,dim(x.combo)[1])
auc <- rep(0,dim(x.combo)[1])
predictors <- rep("null",dim(x.combo)[1])
allcombo.out <- as.data.frame(cbind(aic,auc,predictors))
allcombo.out$aic <- as.numeric(allcombo.out$aic)
allcombo.out$auc <- as.numeric(allcombo.out$auc)
allcombo.out$predictors <- as.character(allcombo.out$predictors)

for (i in 1:dim(x.combo)[1]) {

  x.names <- x.list[x.combo[i,]]
  x <- as.data.frame(df[,x.names])
  names(x) <- x.names

  df.iter <- as.data.frame(cbind(x,binary))

  y.formula <- "binary~"
  x.formula <- paste(x.list[x.combo[i,]],collapse="+")
  iter.formula <- as.formula(paste(y.formula,x.formula,sep=""))

  iter.glm <- glm(iter.formula, data=df.iter, family=binomial)
  iter.prob <- predict.glm(iter.glm, type="response")

  cat(iter.glm$aic,"\n")
  allcombo.out[i,"aic"] <- iter.glm$aic
  allcombo.out[i,"auc"] <- ROC(test=iter.prob,stat=df$binary)$AUC
  allcombo.out[i,"predictors"] <- x.formula


}


# Final step: sort the results from lowest AIC to highest...

allcombo.out <- allcombo.out[sort.list(allcombo.out$aic),]
```

Appendix B

Suggested References

- Bailey, T.C., and A.C. Gatrell. 1995. Interactive spatial data analysis. Prentice Hall, Toronto.

- Bivand, R. 2006. Implementing spatial data analysis software tools in R. Geographical Analysis 38:23-40.
- Chambers, J.M., and T.J. Hastie. 1993. Statistical Models in S. Chapman & Hall, London.
- Fotheringham, A.S., C. Brunsdon, and M. Charlton. 2000. Quantitative geography: perspectives on spatial data analysis. Sage Publications, Ltd. London.
- Fotheringham, A.S., C. Brunsdon, and M. Charlton. 2002. Geographically weighted regression: the analysis of spatially varying relationships. John Wiley & Sons, Ltd.
- Venables, W.N., and B.D. Ripley. Modern Applied Statistics with S. 4th ed. Springer, New York.
- Rey, S.J., and L. Anselin. 2006. Recent advances in software for spatial analysis in the social sciences. Geographical Analysis 38: 1-4.
- Selvin, S. 1998. Modern applied biostatistical methods using S-Plus. Oxford University Press, New York.