In this session, we will

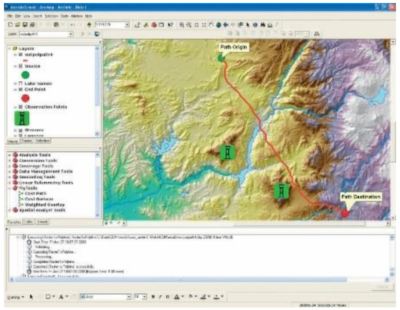
- Go over computer lab logistics and software
- Introduce our practical modeling exercise and the line transect survey data we will use for it
- Discuss strategies for using ArcGIS and R together
- Move our survey sightings from CSV → ArcGIS → R

Software

Our needs

- Explore and manipulate tabular and geospatial data
- Download, visualize, project, and sample gridded environmental data
- Make maps
- Perform general statistical exploration and analysis
- Fit and utilize detection functions
- Fit and utilize generalized additive models (GAMs)

ArcGIS



- First and foremost, a graphical user interface (ArcMap)

\+ Excellent for making maps

\+ Excellent for manipulating spatial data
  - Without programming, via Model Builder diagrams
  - With programming, via Python and other languages

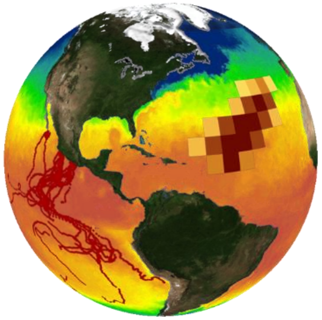
– Poor for statistical analysis or plots, except for specific scenarios, unless you program it yourself

– Has difficulty with scientific data formats (HDF, netCDF, OPeNDAP) and is not very “time-aware”
  - Both of these have been improving with recent releases

– ArcGIS Desktop runs only on Microsoft Windows (currently)

– Closed source, costs a lot of money

Marine Geospatial Ecology Tools (MGET)



- Collection of 300 geoprocessing tools that plugs into ArcGIS
- Can also be invoked from Python
- Requires Windows + ArcGIS
- Free, open source
- Many tools not marine-specific

- Marine Geospatial Ecology Tools
  - Connectivity Analysis
  - Conversion
  - Data Management
  - Data Products
  - Fishery Analysis
  - Oceanographic Analysis
  - Spatial Analysis
  - Statistics

- In this workshop, we will mainly use tools related to acquiring and manipulating environmental data for use in our density modeling exercise

http://mgel.env.duke.edu/mget (or Google "MGET")

R

- First and foremost, a programming language

+ Cross platform, open source, free (as in freedom)

+ Excellent for statistical analysis and plots

+ Excellent for manipulating tabular data
  - Once you get the data loaded into R

± Excellent for manipulating raster data, less so for vector

– High learning curve, even for seasoned programmers

– Very tedious for making maps, relative to GIS software
  - But can produce excellent results, with programming

Distance R packages

- R packages for distance sampling include:
  - ***mrds*** - fits detection functions to point and line transect distance sampling survey data, for both single and double observer surveys.
  - ***Distance*** - a simpler interface to *mrds* for single observer distance sampling surveys.
  - ***dsm*** - fits density surface models to spatially-referenced distance sampling data. Count data are corrected using detection functions fitted using *mrds* or *Distance*. Spatial models are constructed using generalized additive models.
- We will spend much of our time with these

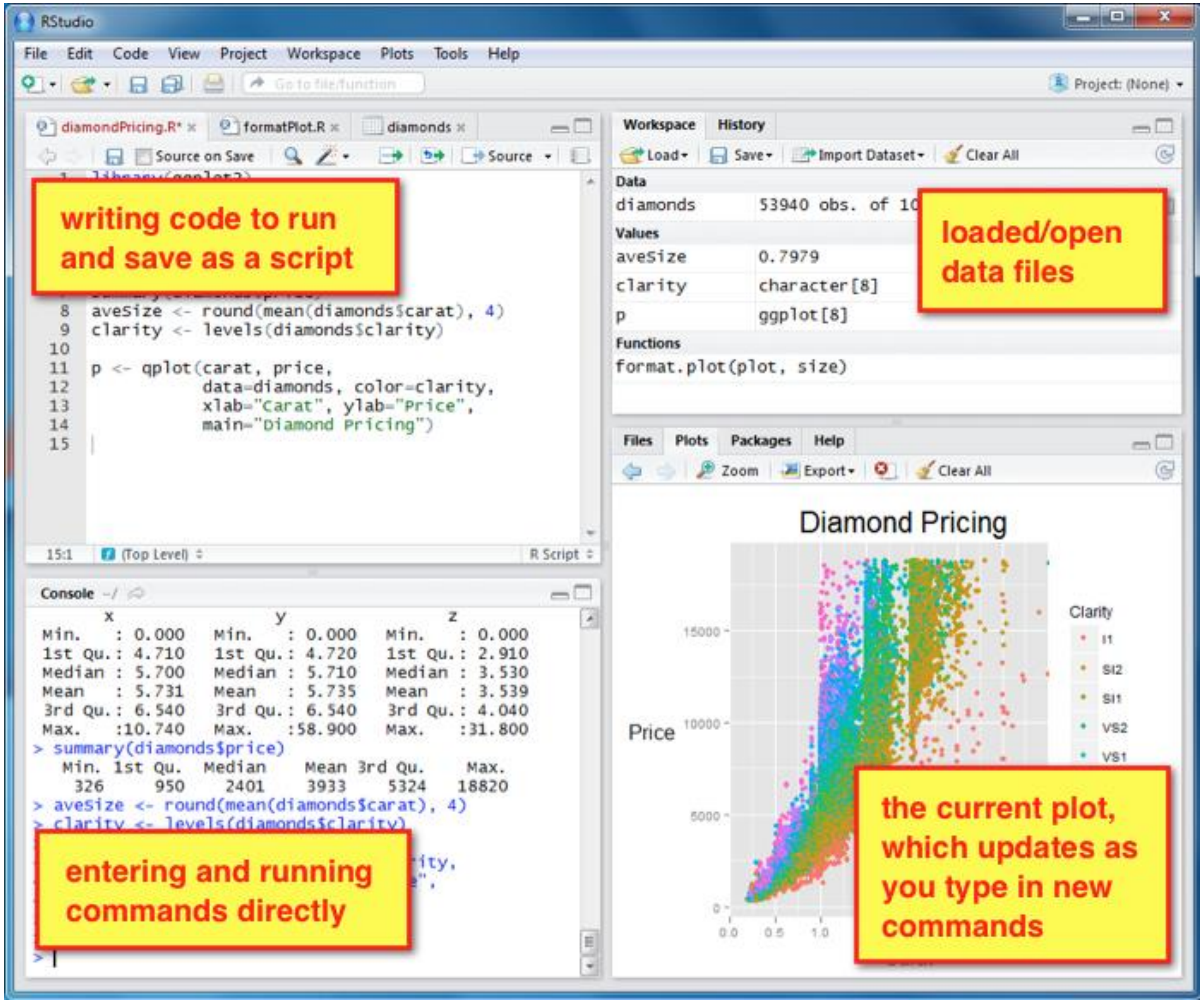
http://distancesampling.org

Other R packages

- ***mgcv*** – for fitting generalized additive models (GAMs). We will spend a lot of time with this package, although functions from *Distance* and *dsm* will wrap it for us.
- ***rgdal, raster*** – for reading and writing geospatial data
- ***ggplot2, viridis*** – for nice plots
- **plyr, reshape2** – for manipulating tabular data, especially R data.frames

RStudio Desktop

- Powerful integrated development environment for R
- Free, open source



```r
library(ggplot2)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

writing code to run and save as a script

Data

diamonds 53940 obs. of 10

Values

aveSize 0.7979

clarity character[8]

p ggplot[8]

Functions

format.plot(plot, size)

loaded/open data files

```
       x                 y                 z         
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
 Median : 5.700   Median : 5.710   Median : 3.530  
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
 Max.   :10.740   Max.   :58.900   Max.   :31.800  
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    326     950    2401    3933    5324   18820 
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
```

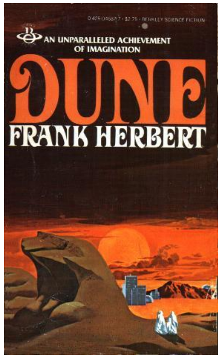
entering and running commands directly

the current plot, which updates as you type in new commands

Image: http://www.rstudio.com and http://clasticdetritus.com

“The people I distrust most are those who want to improve our lives but have only one course of action.”

— Frank Herbert



Computer lab software setup

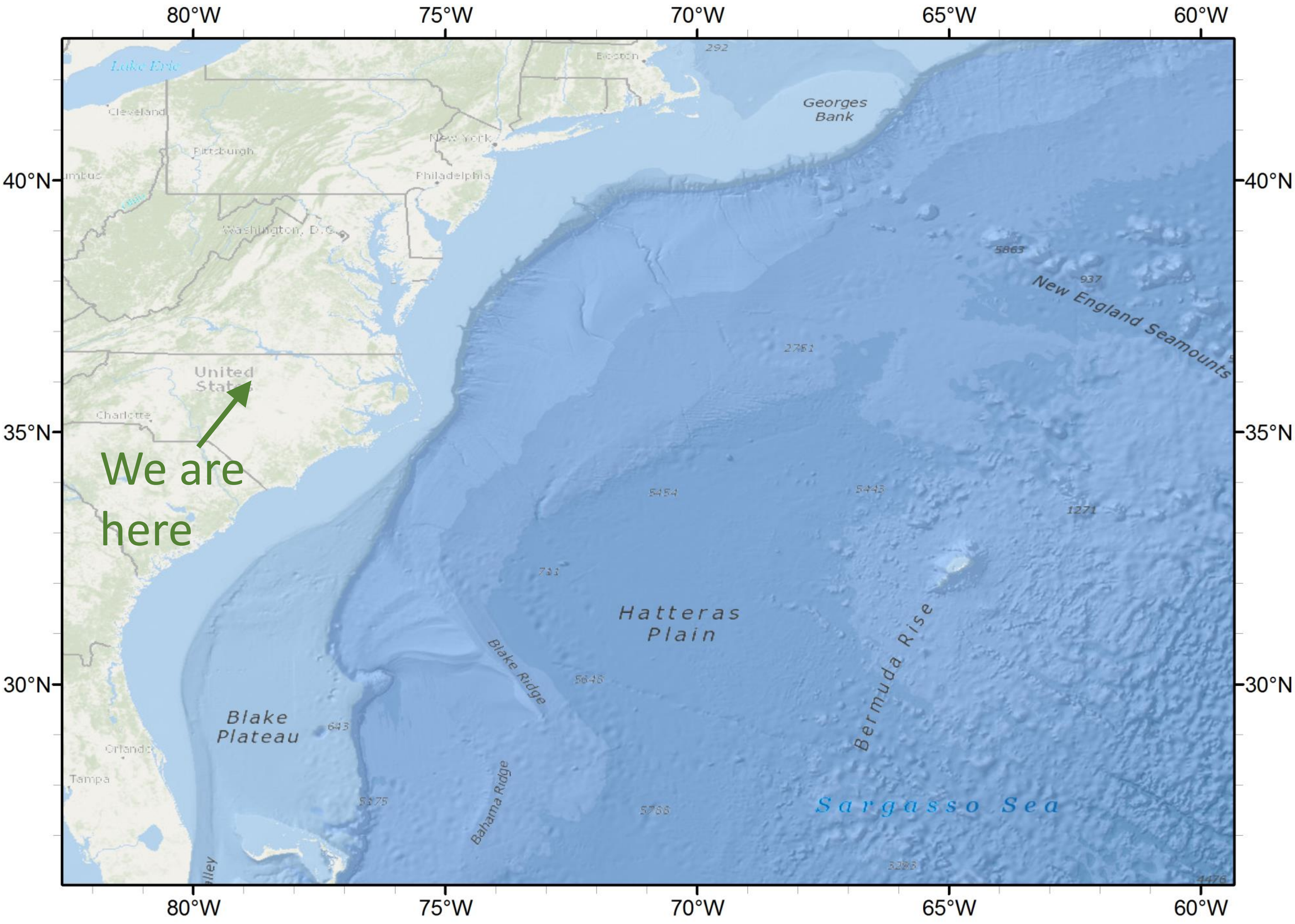
1. In your browser, open

   http://distancesampling.org/workshops/duke-spatial-2015/

2. Go to **Course Materials** and click on **Slides**

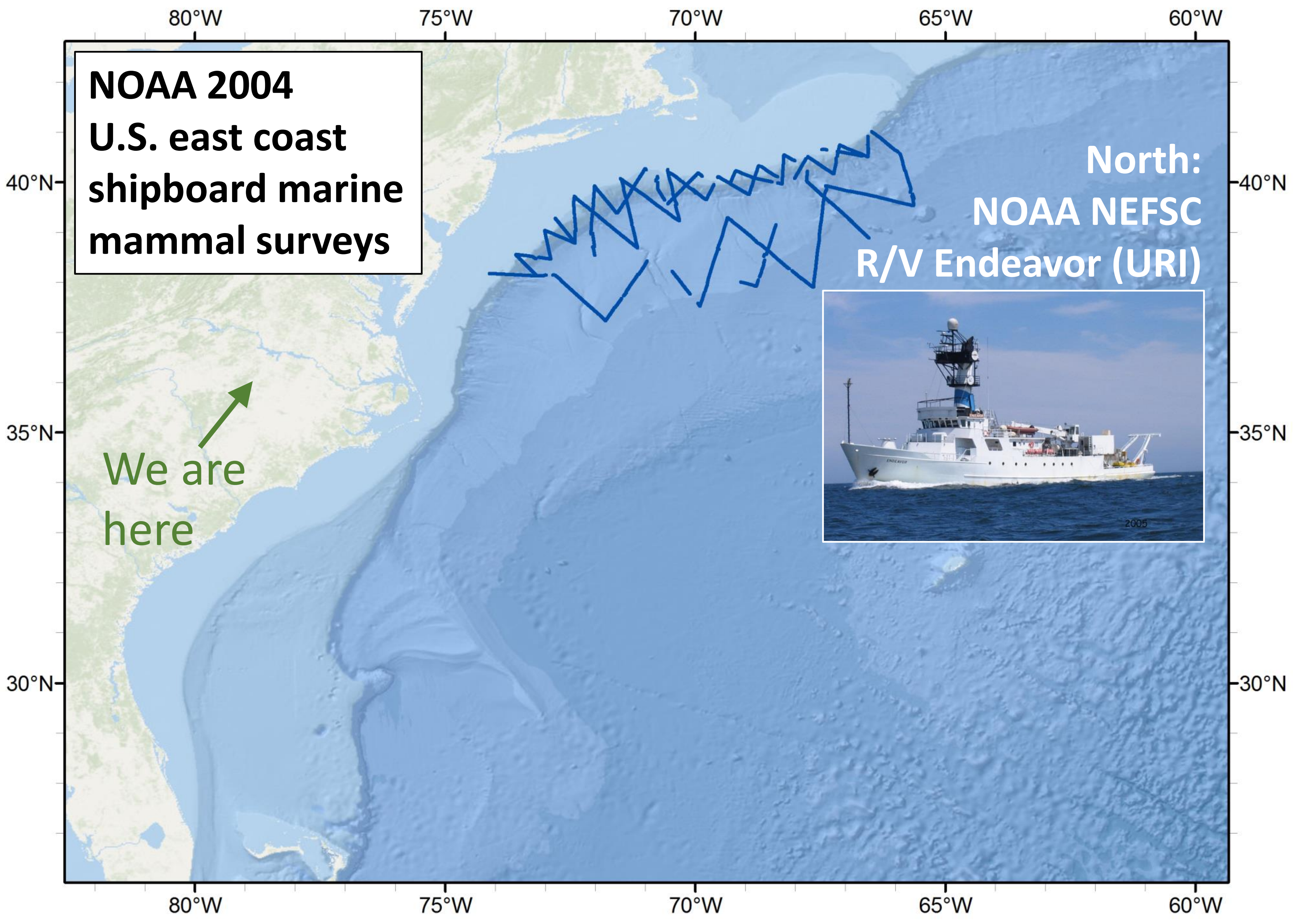
3. Open the **Software Setup** PDF and follow the instructions

Practical modeling exercise



80°W
75°W
70°W
65°W
60°W
40°N
35°N
30°N
Lake Erie
Cleveland
Pittsburgh
Boston
New York
Philadelphia
Washington, D.C.
United States
Charlotte
Orlando
Tampa
Georges Bank
292
5863
937
New England Seamounts
2751
5454
5443
1271
711
Hatteras Plain
Blake Ridge
5646
Bermuda Rise
Blake Plateau
643
5175
Bahama Ridge
5788
Sargasso Sea
3283
4476

We are here

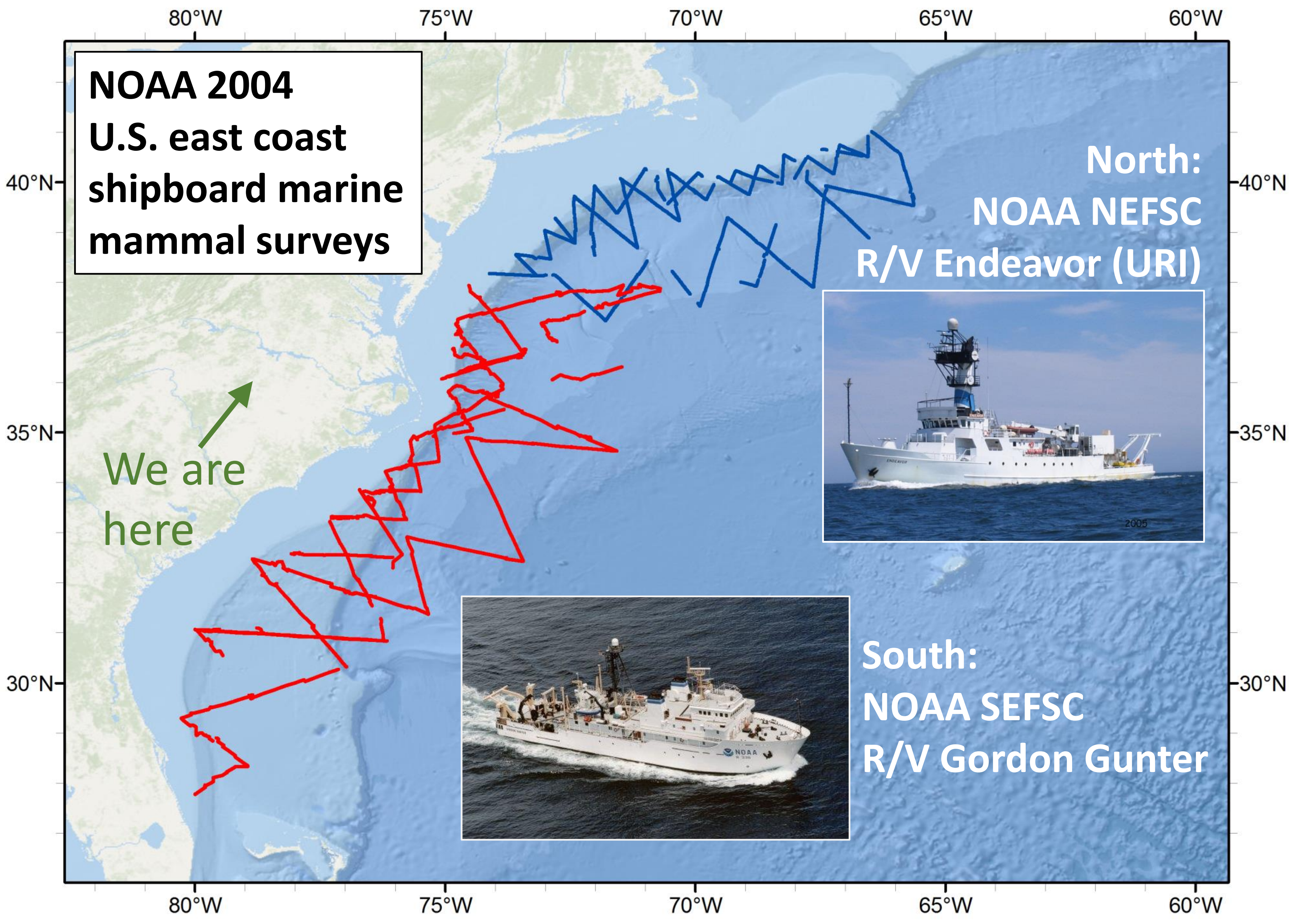


NOAA 2004
U.S. east coast
shipboard marine
mammal surveys

North:
NOAA NEFSC
R/V Endeavor (URI)

We are here

NOAA 2004 U.S. east coast shipboard marine mammal surveys



North:
NOAA NEFSC
R/V Endeavor (URI)

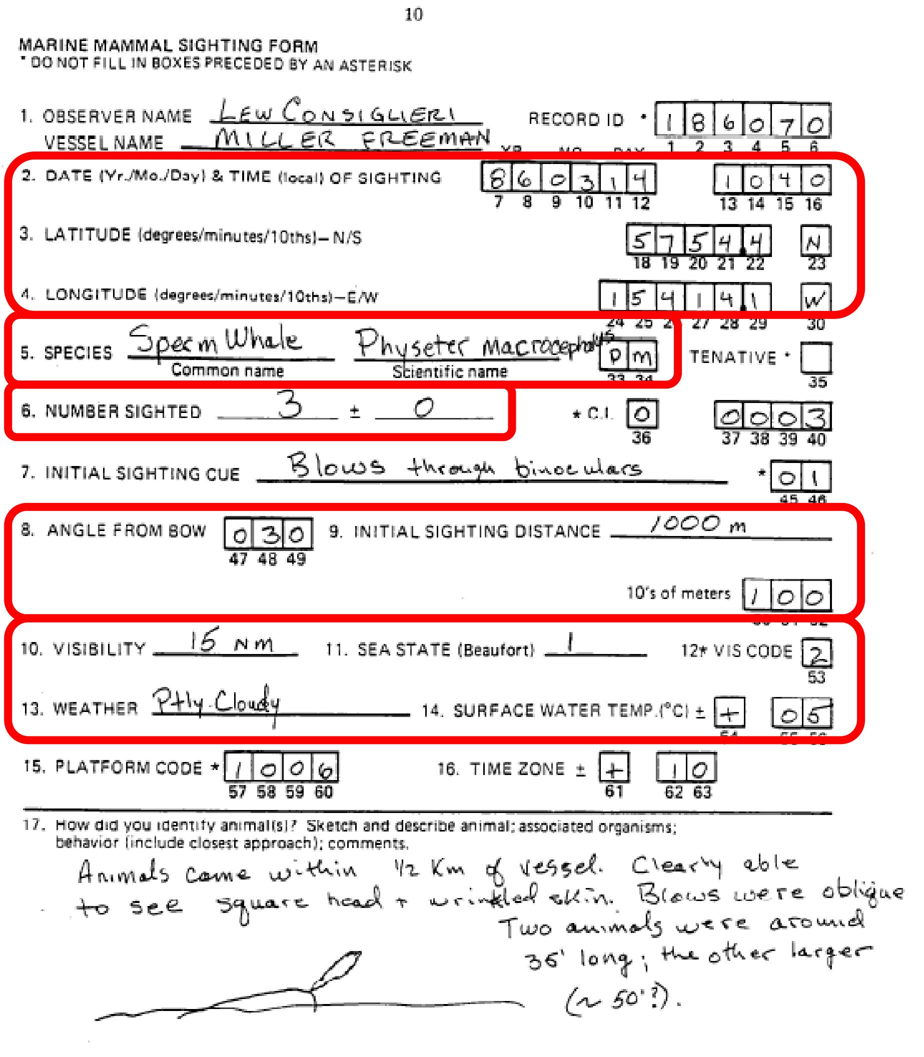
South:
NOAA SEFSC
R/V Gordon Gunter

Observers on the R/V Gordon Gunter

Observer team

Observers on the R/V Gordon Gunter

Photo:
Kimberly Gogan



10

MARINE MAMMAL SIGHTING FORM
* DO NOT FILL IN BOXES PRECEDED BY AN ASTERISK

1. OBSERVER NAME Lew Consiglieri RECORD ID * 1 8 6 0 7 0
1 2 3 4 5 6
VESSEL NAME Miller Freeman

2. DATE (Yr./Mo./Day) & TIME (local) OF SIGHTING 8 6 0 3 1 4 1 0 4 0
7 8 9 10 11 12 13 14 15 16

3. LATITUDE (degrees/minutes/10ths)—N/S 5 7 5 4 4 N
18 19 20 21 22 23

4. LONGITUDE (degrees/minutes/10ths)—E/W 1 5 4 1 4 1 W
24 25 2 27 28 29 30

5. SPECIES Sperm Whale (Common name) Physeter macrocephalus (Scientific name) P M TENATIVE * ☐
35

6. NUMBER SIGHTED 3 ± 0 * C.I. 0 0 0 0 3
36 37 38 39 40

7. INITIAL SIGHTING CUE Blows through binoculars * 0 1
45 46

8. ANGLE FROM BOW 0 3 0
47 48 49

9. INITIAL SIGHTING DISTANCE 1000 m

10's of meters 1 0 0

10. VISIBILITY 15 NM 11. SEA STATE (Beaufort) 1 12* VIS CODE 2
53

13. WEATHER Ptly. Cloudy 14. SURFACE WATER TEMP.(°C) ± + 0 5

15. PLATFORM CODE * 1 0 0 6 16. TIME ZONE ± + 1 0
57 58 59 60 61 62 63

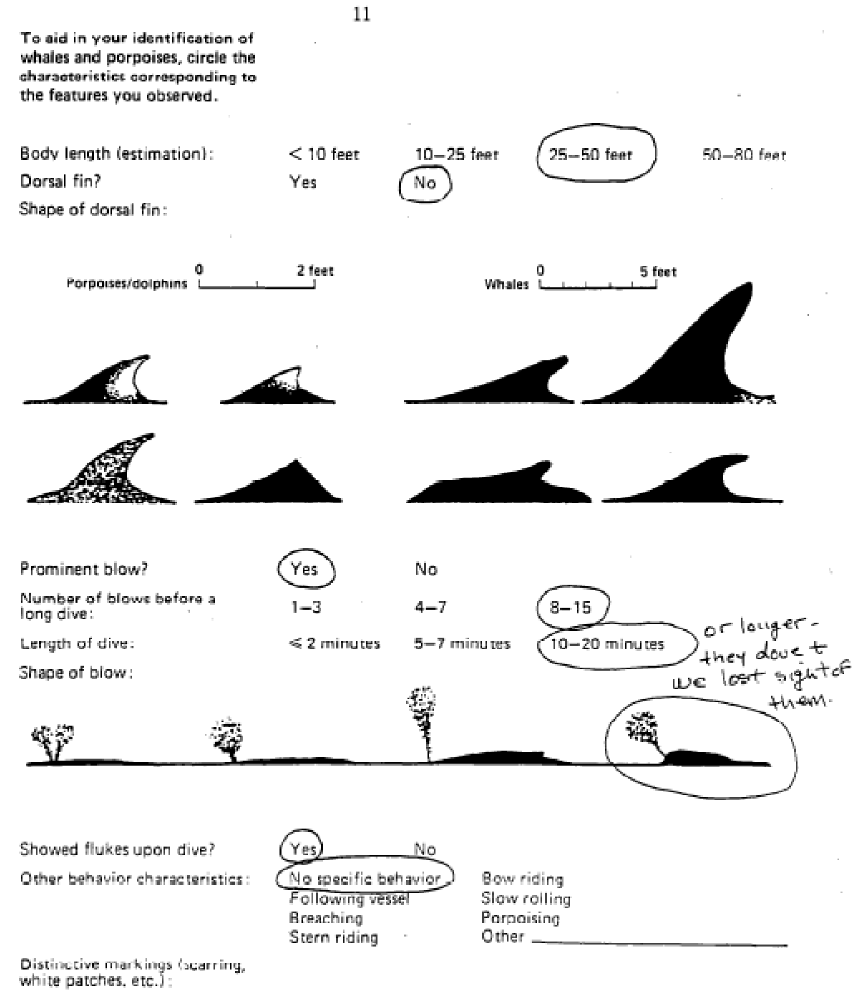
17. How did you identify animal(s)? Sketch and describe animal; associated organisms; behavior (include closest approach); comments.

Animals came within ½ Km of vessel. Clearly able to see square head + wrinkled skin. Blows were oblique

Two animals were around 36' long; the other larger (~ 50'?).

Figure 1.--Marine mammal sighting form (front).

65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80



11

To aid in your identification of whales and porpoises, circle the characteristics corresponding to the features you observed.

Body length (estimation): < 10 feet 10–25 feet [✓ 25–50 feet] 50–80 feet
Dorsal fin? Yes [✓ No]
Shape of dorsal fin:

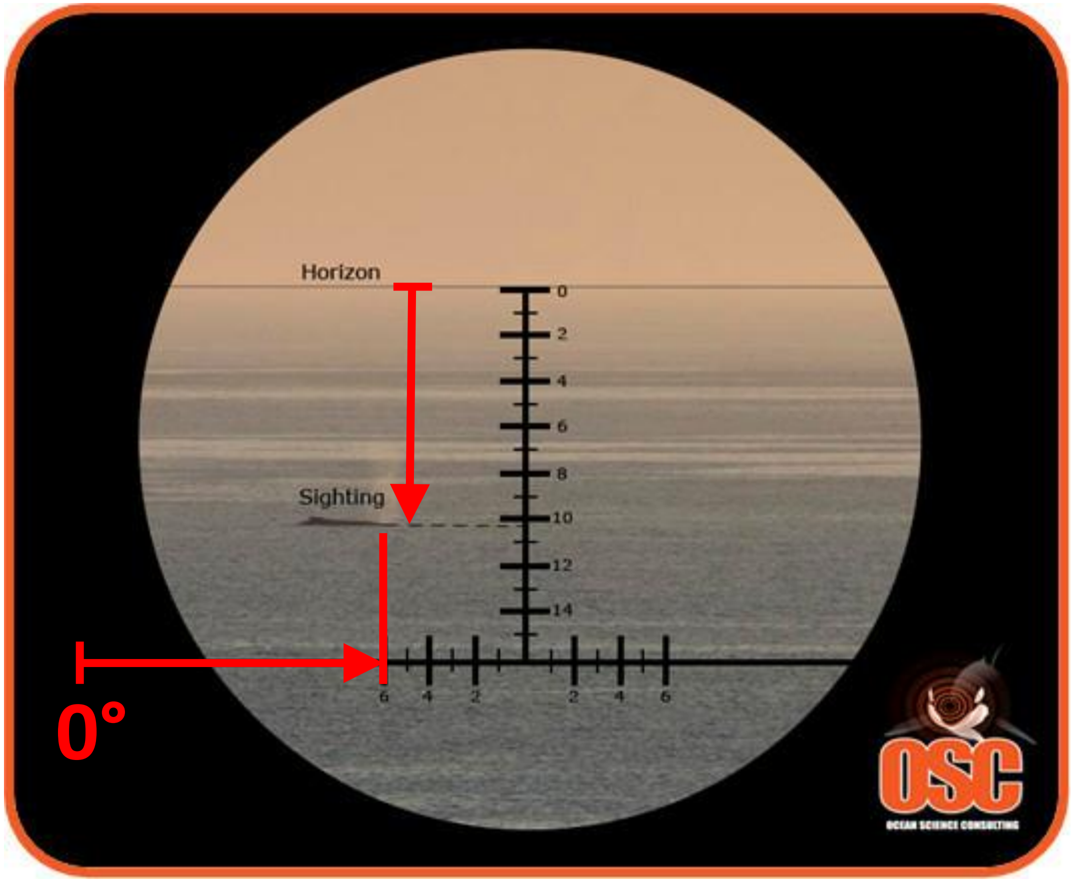
Prominent blow? [✓ Yes] No
Number of blows before a long dive: 1–3 4–7 [✓ 8–15]
Length of dive: ≤ 2 minutes 5–7 minutes [✓ 10–20 minutes] or longer - they dove + we lost sight of them.
Shape of blow:

Showed flukes upon dive? [✓ Yes] No
Other behavior characteristics: [✓ No specific behavior] Following vessel Breaching Stern riding Bow riding Slow rolling Porpoising Other
Distinctive markings (scarring, white patches, etc.):

Figure 2.--Marine mammal sighting form (back).

Boucher CG, Boaz CJ (1989) Documentation for the Marine Mammal Sightings Database of the National Marine Mammal Laboratory. NOAA Technical Memorandum NMFS F/NWC-159. 60 p.

Perpendicular distances to sightings using binocular reticles



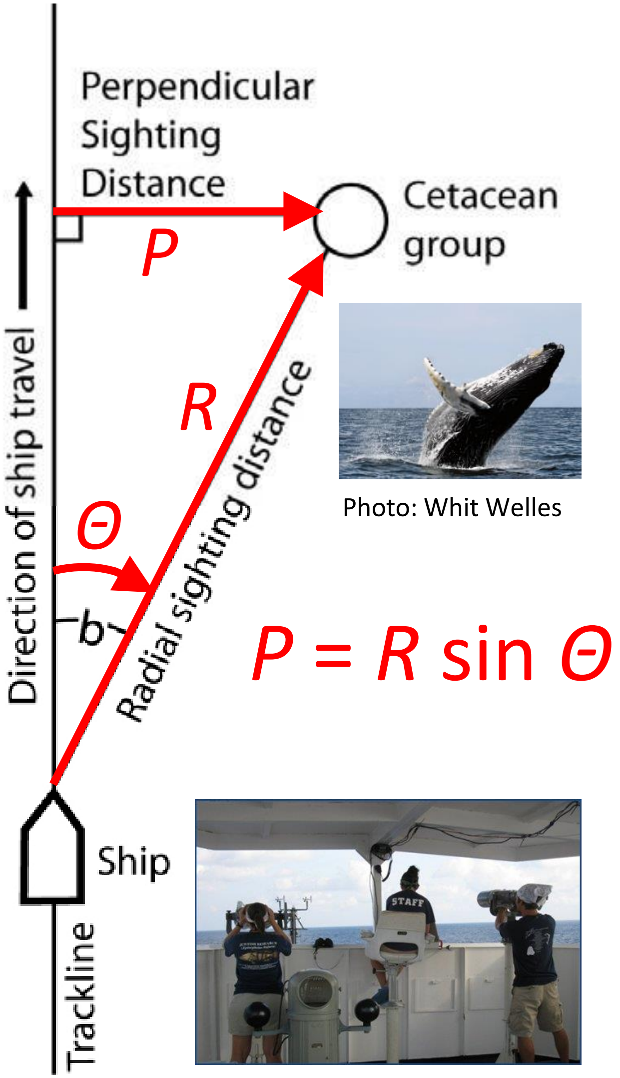


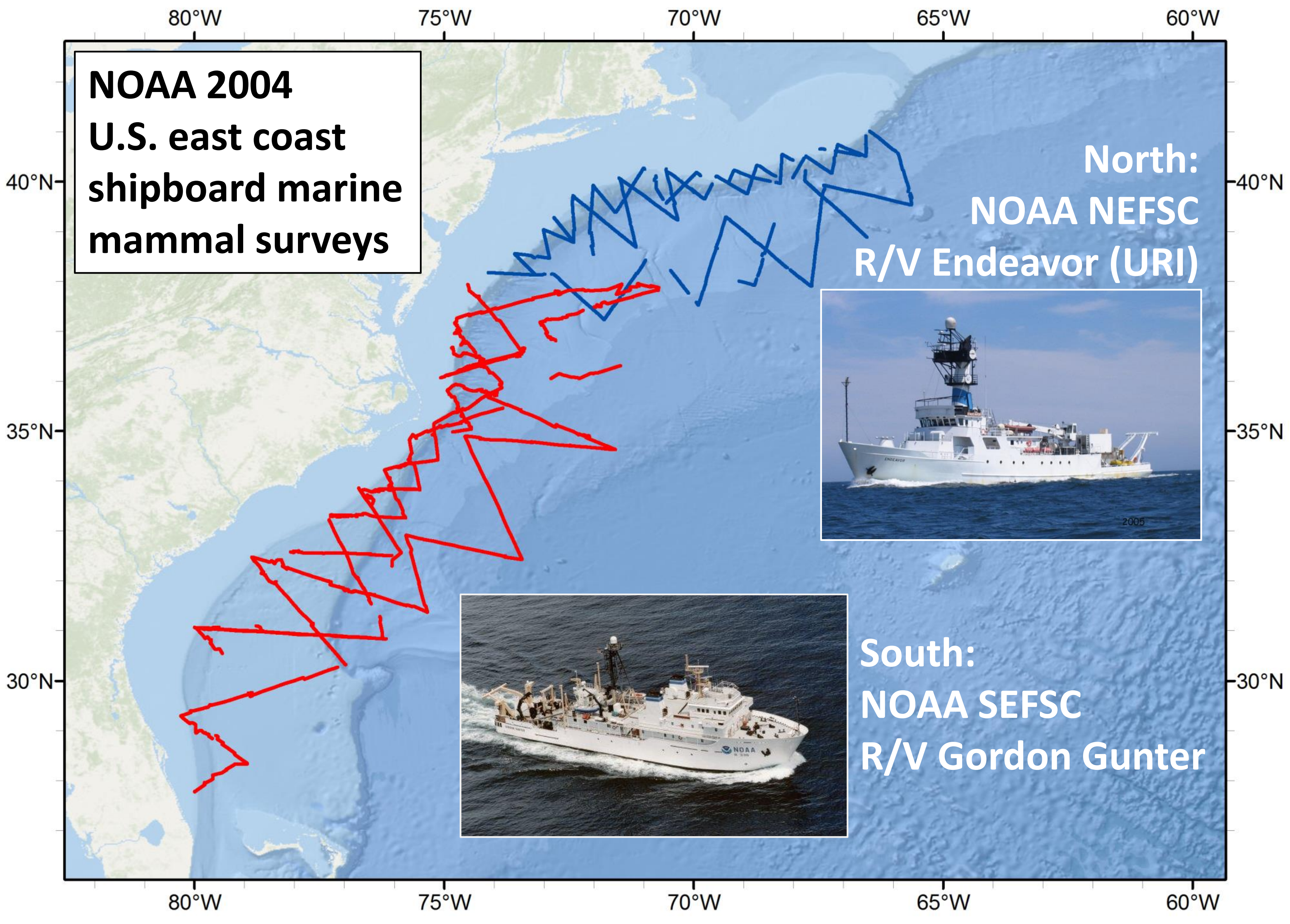
Photo: Whit Welles

$$P = R \sin \Theta$$

Our species of interest:
Sperm whale
Physeter macrocephalus

Photo: Franco Banfi

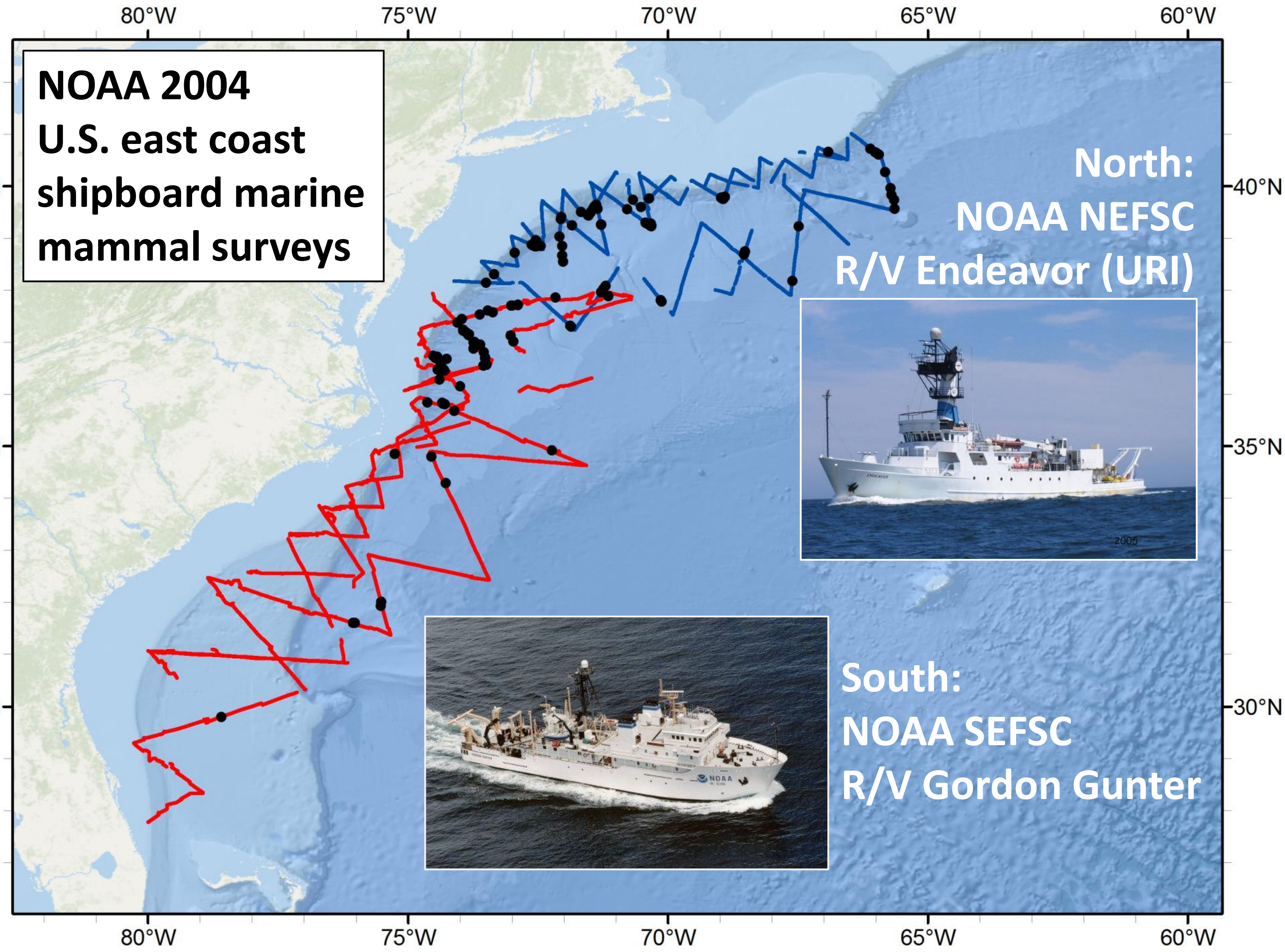
NOAA 2004
U.S. east coast
shipboard marine
mammal surveys



North:
NOAA NEFSC
R/V Endeavor (URI)

South:
NOAA SEFSC
R/V Gordon Gunter

NOAA 2004
U.S. east coast shipboard marine mammal surveys



North:
NOAA NEFSC
R/V Endeavor (URI)

South:
NOAA SEFSC
R/V Gordon Gunter

NOAA’s abundance estimates (Waring et al. 2007):

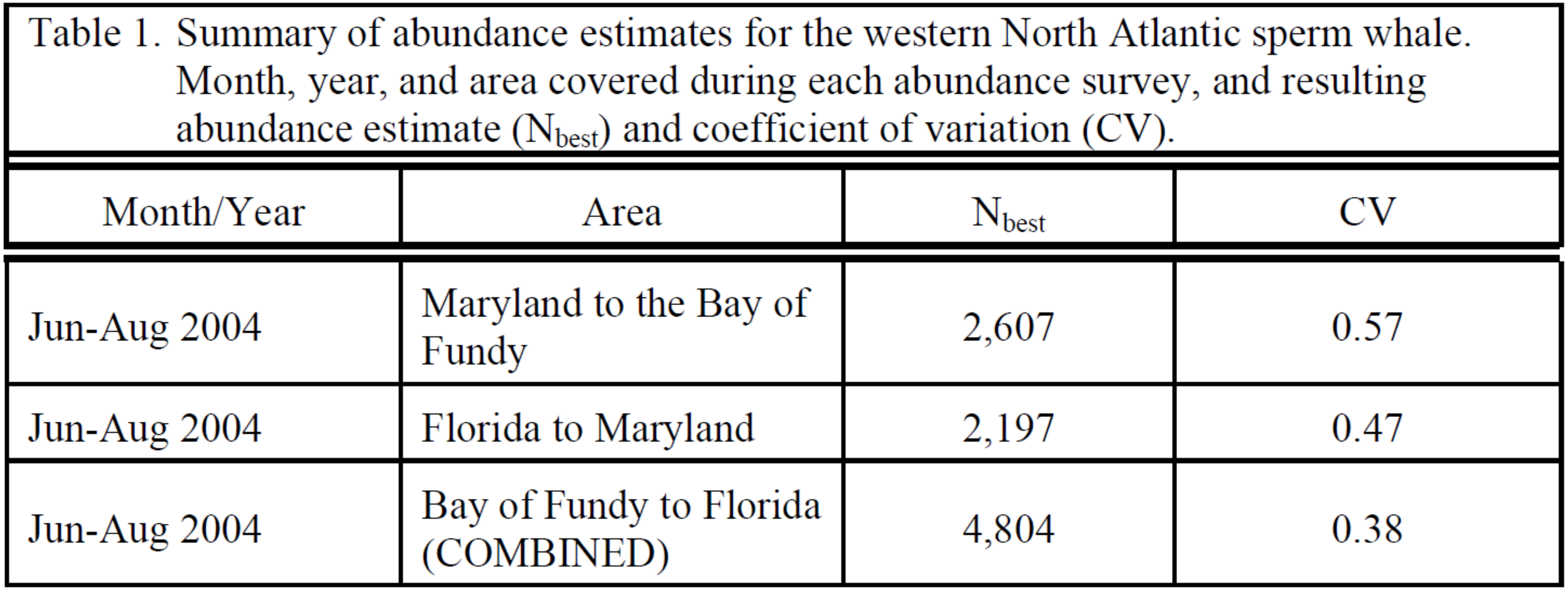


Table 1. Summary of abundance estimates for the western North Atlantic sperm whale. Month, year, and area covered during each abundance survey, and resulting abundance estimate (N_{best}) and coefficient of variation (CV).

Month/Year	Area	N_{best}	CV
Jun-Aug 2004	Maryland to the Bay of Fundy	2,607	0.57
Jun-Aug 2004	Florida to Maryland	2,197	0.47
Jun-Aug 2004	Bay of Fundy to Florida (COMBINED)	4,804	0.38

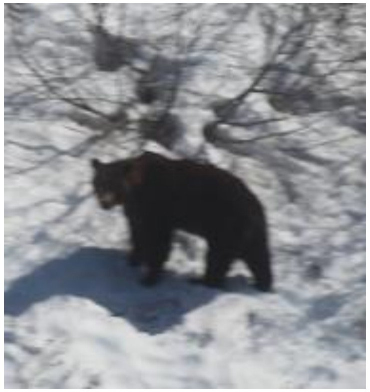
Waring GT, Josephson E, Fairfield-Walsh CP, Maze-Foley K (2007) U.S. Atlantic and Gulf of Mexico Marine Mammal Stock Assessments -- 2007. NOAA Tech Memo NMFS NE 205. 415 p.

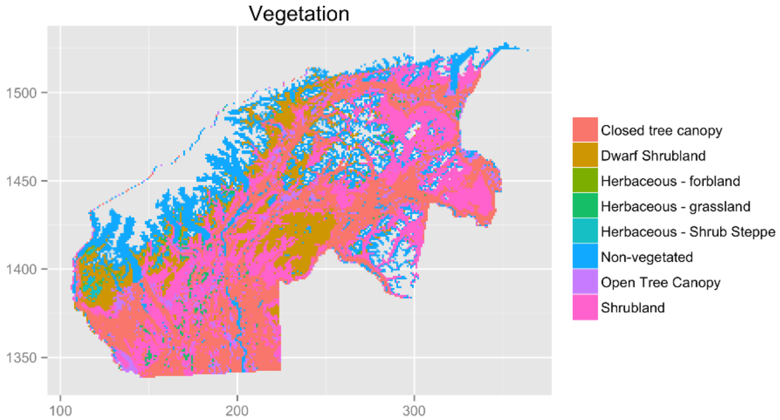
Our goals:

- Produce our own abundance estimates from NOAA’s data
- Go beyond this: produce a density surface (animals km^{-2})

This methodology is generic!

- We’re teaching a marine example because one of us works mainly on marine species
- The methodology and most of the tools are generic
- If you are a terrestrial ecologist, please feel free to speak up, raise terrestrial questions and examples, and represent land-dwellers with pride!





Photos and figure: David L Miller and colleagues

Let’s explore the data…

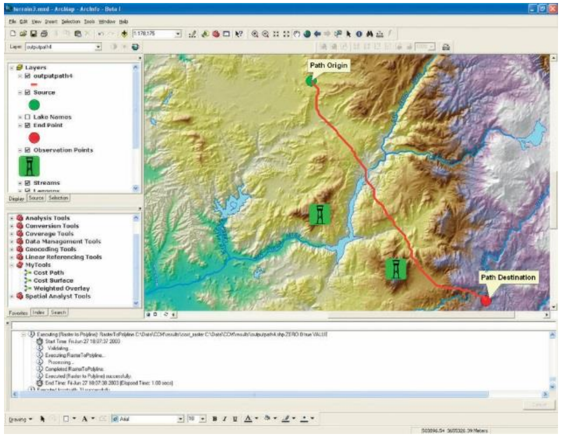
Using ArcGIS and R together

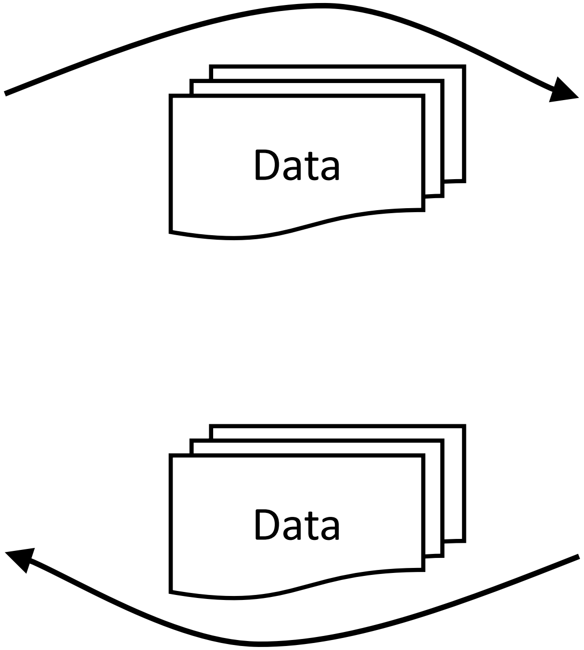
Two main approaches

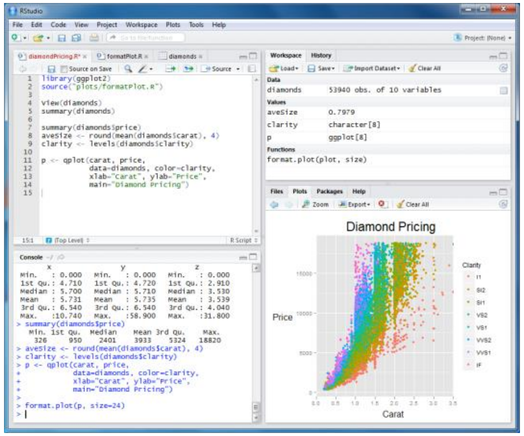
- ***Exchange data -*** run both programs interactively and manually move data back and forth between them
  - We will do this in our workshop
- ***Automation*** - execute one program from within the other, or both from a third program, to coordinate their execution from an automated workflow
  - We will not do this, but I can discuss it at the end of the session, if there is time and interest

Exchanging data by writing files

ArcGIS writes, R reads







```
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
Data
diamonds        53940 obs. of 10 variables
Values
aveSize         0.7979
clarity         character[8]
p               ggplot[8]
Functions
format.plot(plot, size)
```

```
       x                 y                 z
Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median : 5.700   Median : 5.710   Median : 3.530
Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> 
```

R writes, ArcGIS reads

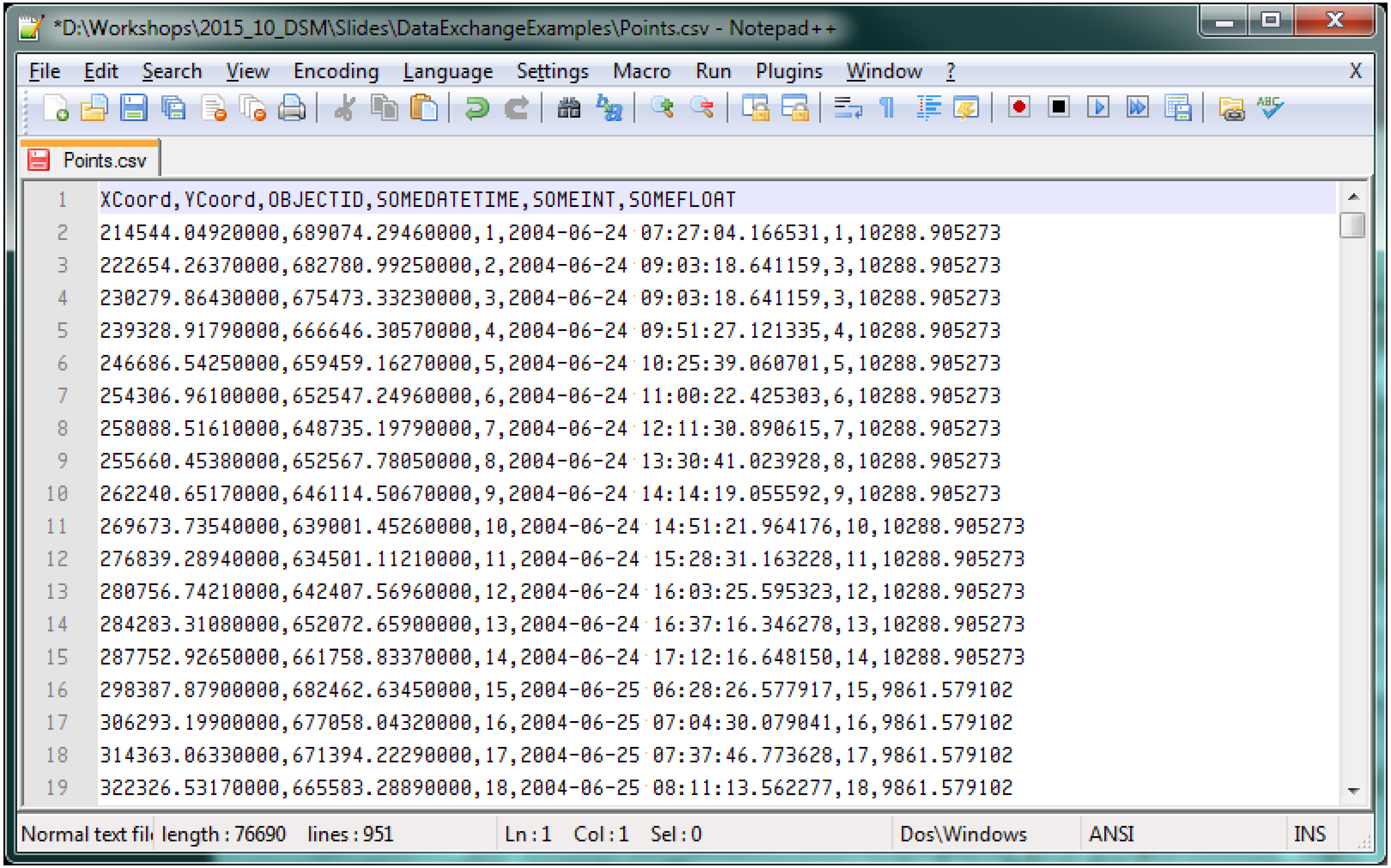
Formats for exchanging data

For tabular data—tables and feature classes in ArcGIS—there are several common alternatives:

- Comma-separated values (CSV) files
- DBF files and shapefiles
- Personal and file geodatabases

For rasters, you can leave them in the formats you already use in ArcGIS (GeoTIFF, IMG, etc.)

Comma-separated values (CSV) files

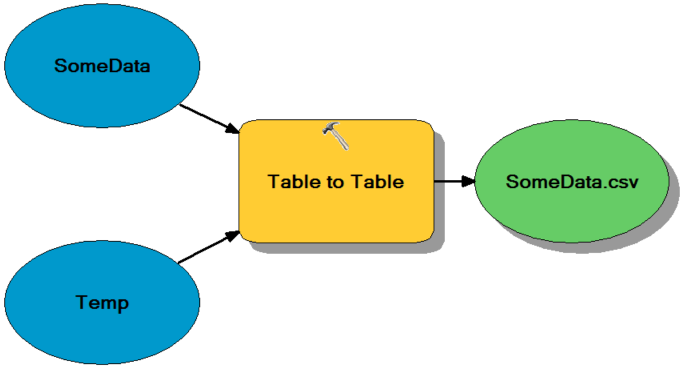


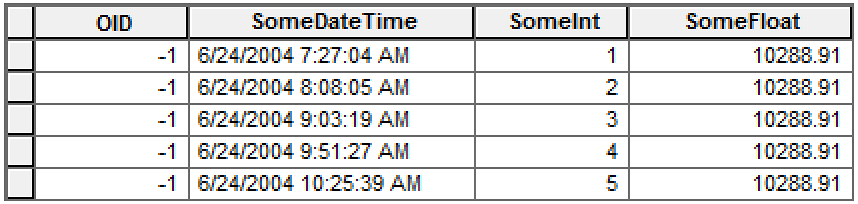
```
XCoord,YCoord,OBJECTID,SOMEDATETIME,SOMEINT,SOMEFLOAT
214544.04920000,689074.29460000,1,2004-06-24 07:27:04.166531,1,10288.905273
222654.26370000,682780.99250000,2,2004-06-24 09:03:18.641159,3,10288.905273
230279.86430000,675473.33230000,3,2004-06-24 09:03:18.641159,3,10288.905273
239328.91790000,666646.30570000,4,2004-06-24 09:51:27.121335,4,10288.905273
246686.54250000,659459.16270000,5,2004-06-24 10:25:39.060701,5,10288.905273
254306.96100000,652547.24960000,6,2004-06-24 11:00:22.425303,6,10288.905273
258088.51610000,648735.19790000,7,2004-06-24 12:11:30.890615,7,10288.905273
255660.45380000,652567.78050000,8,2004-06-24 13:30:41.023928,8,10288.905273
262240.65170000,646114.50670000,9,2004-06-24 14:14:19.055592,9,10288.905273
269673.73540000,639001.45260000,10,2004-06-24 14:51:21.964176,10,10288.905273
276839.28940000,634501.11210000,11,2004-06-24 15:28:31.163228,11,10288.905273
280756.74210000,642407.56960000,12,2004-06-24 16:03:25.595323,12,10288.905273
284283.31080000,652072.65900000,13,2004-06-24 16:37:16.346278,13,10288.905273
287752.92650000,661758.83370000,14,2004-06-24 17:12:16.648150,14,10288.905273
298387.87900000,682462.63450000,15,2004-06-25 06:28:26.577917,15,9861.579102
306293.19900000,677058.04320000,16,2004-06-25 07:04:30.079041,16,9861.579102
314363.06330000,671394.22290000,17,2004-06-25 07:37:46.773628,17,9861.579102
322326.53170000,665583.28890000,18,2004-06-25 08:11:13.562277,18,9861.579102
```

CSV files for tables

- Just text; no way to specify data types of columns
- Due to that and other limitations of ArcGIS, CSV is not an appropriate default format when using ArcGIS
- Export from ArcGIS messes up certain columns

Send a table from ArcGIS to R with a CSV:





OID	SomeDateTime	SomeInt	SomeFloat
-1	6/24/2004 7:27:04 AM	1	10288.91
-1	6/24/2004 8:08:05 AM	2	10288.91
-1	6/24/2004 9:03:19 AM	3	10288.91
-1	6/24/2004 9:51:27 AM	4	10288.91
-1	6/24/2004 10:25:39 AM	5	10288.91

All OBJECTIDs set to -1

```
> somedata <- read.csv("C:/Temp/SomeData.csv", stringsAsFactors=FALSE)
```

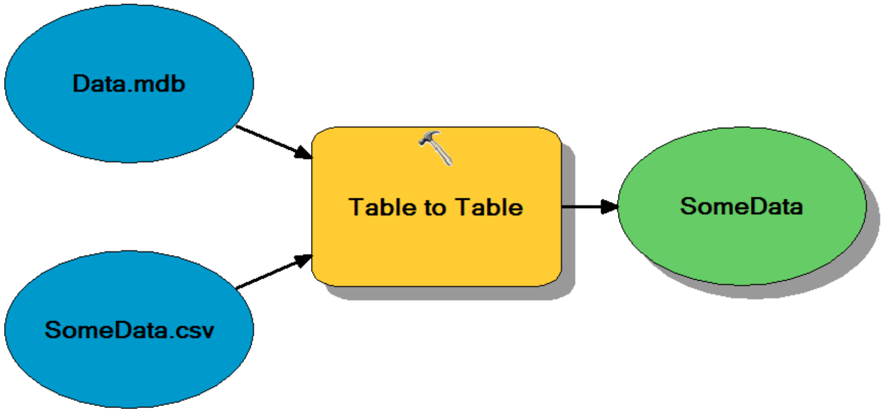
For date columns, use `colClasses` parameter to specify data type

CSV files for tables

Send a table from R to ArcGIS with a CSV:

```
> write.csv(somedata, "C:/Temp/SomeData.csv", row.names=FALSE, na="")
```

CSVs may be used directly in ArcGIS for certain tasks. But often it is necessary to convert them to more structured format, such as a geodatabase table or DBF file:



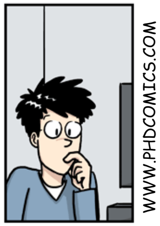
CSV files for feature classes

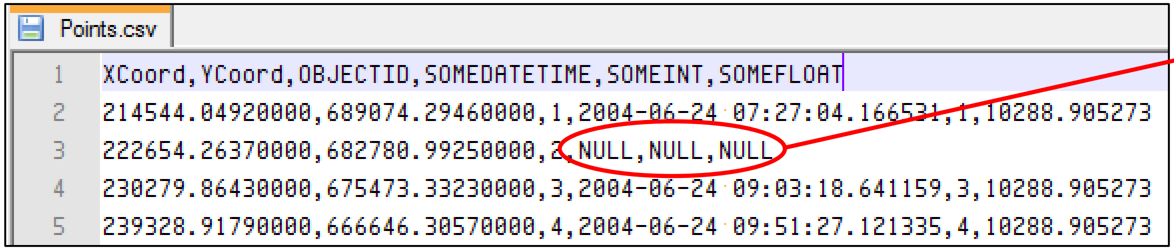
– Same limitations as with tables

– Cannot easily handle geometries other than points

Send points from ArcGIS to R with a CSV:

From the Spatial Stats toolbox!?





```
XCoord,YCoord,OBJECTID,SOMEDATETIME,SOMEINT,SOMEFLOAT
214544.04920000,689074.29460000,1,2004-06-24 07:27:04.166531,1,10288.905273
222654.26370000,682780.99250000,2,NULL,NULL,NULL
230279.86430000,675473.33230000,3,2004-06-24 09:03:18.641159,3,10288.905273
239328.91790000,666646.30570000,4,2004-06-24 09:51:27.121335,4,10288.905273
```

NULL values written as "NULL"; R converts column to character data type!

```
> points <- read.csv("C:/Temp/Points.csv", stringsAsFactors=FALSE)
```

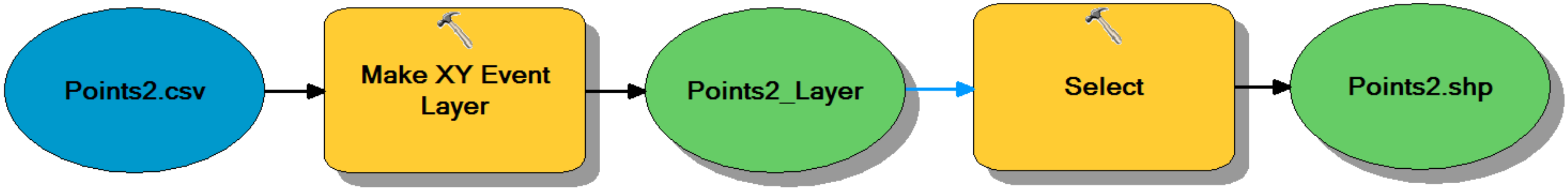
For date columns, use `colClasses` parameter to specify data type

CSV files for feature classes

Send points from R to ArcGIS with a CSV:

```
> write.csv(points, "D:/Temp/Points2.csv", row.names=FALSE, na="")
```

Make sure `points` has columns for x and y coordinates



Makes an in-memory feature layer

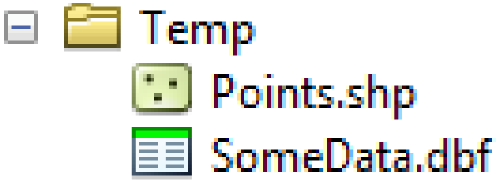
Only needed if you wish to save the layer

DBF files for tables

+ Suitable as default format in ArcGIS, but:

– Significant limitations: 10 char column names; date fields do not have times; little support for NULL values

Read a DBF file into R:



Temp
Points.shp
SomeData.dbf

```
> library(foreign)
> somedata <- read.dbf("C:/Temp/SomeData.dbf", as.is=TRUE)
```

Write a DBF file from R:

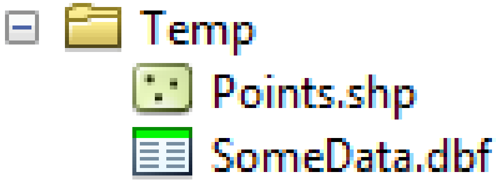
```
> write.dbf(somedata, "C:/Temp/SomeData2.dbf", factor2char=TRUE)
```

Shapefiles for vector data

+ Suitable as default format in ArcGIS

– Same limitations as DBF: 10 char column names; date fields do not have times; little support for NULL values

Read a shapefile into R:



Temp
Points.shp
SomeData.dbf

For DATE columns, readOGR creates a character column in the returned data.frame. We must parse it, e.g. using as.POSIXct().

```
> library(rgdal)
> points <- readOGR("D:/Temp", "Points", stringsAsFactors=FALSE)
> points$SomeDateTime <- as.POSIXct(points$SomeDateTime)
```

Write a shapefile from R:

```
> writeOGR(points, "D:/Temp", "Points", driver="ESRI Shapefile")
```

For POSIXct (etc.) columns, writeOGR creates a TEXT column in the shapefile.

Personal and file geodatabases

+ Multiple tables and feature classes in single file or dir.
+ Avoids archaic limitations of CSV, DBF, and shapefile
– Different R packages needed depending on scenario

Personal geodatabase (.mdb file)

± MS Access format; can open in many tools; can be hard on Linux

– Total file size limited to 2 GB

– ESRI is depreciating this format

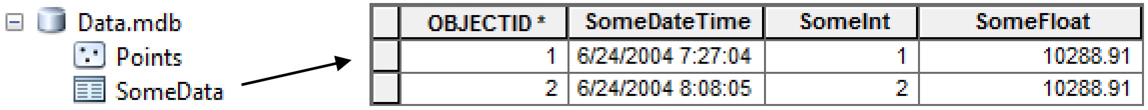
File geodatabase (.gdb directory)

+ No size limitation

– Proprietary ESRI format; limited interoperability

With the RODBC package:

Read a table from a personal GDB (or other Access DB):



Data.mdb
Points
SomeData

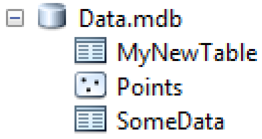
	OBJECTID *	SomeDateTime	SomeInt	SomeFloat
	1	6/24/2004 7:27:04	1	10288.91
	2	6/24/2004 8:08:05	2	10288.91

```
> library(RODBC)     # May not be available on all Linux distros
> conn <- odbcConnectAccess("D:/Temp/Data.mdb")    # odbcConnect on Linux
> data <- sqlQuery(conn, "SELECT * FROM SomeData", stringsAsFactors=FALSE)
> close(conn)
```

Neither works with file GDBs!

Write a table to a personal GDB (or other Access DB):

```
> library(RODBC)
> conn <- odbcConnectAccess("D:/Temp/Data.mdb")
> sqlWrite(conn, data, "MyNewTable", rownames=FALSE,
           varTypes=c(SomeDateTime="datetime"))
> close(conn)
```

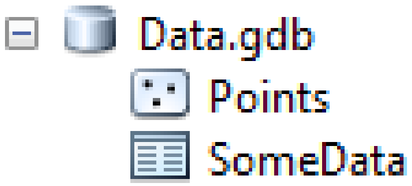


Data.mdb
MyNewTable
Points
SomeData

Necessary for ArcGIS to add or recognize the table's OBJECTID

With the rgdal package:

Read a feature class from a personal or file GDB:



Data.gdb
Points
SomeData

As with shapefiles, for DATE columns, readOGR creates a character column in the returned data.frame. Must parse, e.g. using as.POSIXct().

```
> library(rgdal)
> points <- readOGR("D:/Temp/Data.gdb", "Points", stringsAsFactors=FALSE)
> points$SomeDateTime <- as.POSIXct(points$SomeDateTime)
```

- You cannot write to geodatabases with rgdal at this time
- In the future, it may be possible to write to file geodatabaseses if some technical and licensing issues are worked out on CRAN (but this looks pretty unlikely)

ESRI’s new initiative

Welcome to the R – ArcGIS Community

Combine the power of ArcGIS and R to solve your spatial problems

The R – ArcGIS Community is a community driven collection of free, open source projects making it easier and faster for R users to work with ArcGIS data, and ArcGIS users to leverage the analysis capabilities of R.

Need the R Statistical Software? Download it now.

https://r-arcgis.github.io/

R-bridge for ArcGIS

- Enables R to read and write any tables or feature classes that are accessible through ArcGIS
- Brand new: July 2015
- Requires ArcGIS 10.3.1+, R 3.1.0+, MS Windows
- Requires administrator rights to install
  - Instructions: https://github.com/R-ArcGIS/r-bridge-install
- Installs the arcgisbinding R library
  - Cannot be installed from CRAN (at least right now)
  - Only works if ArcGIS is installed; checks your license
  - Core implemented with C++, COM, ATL, ArcObjects
  - Open source (!) Apache License 2.0

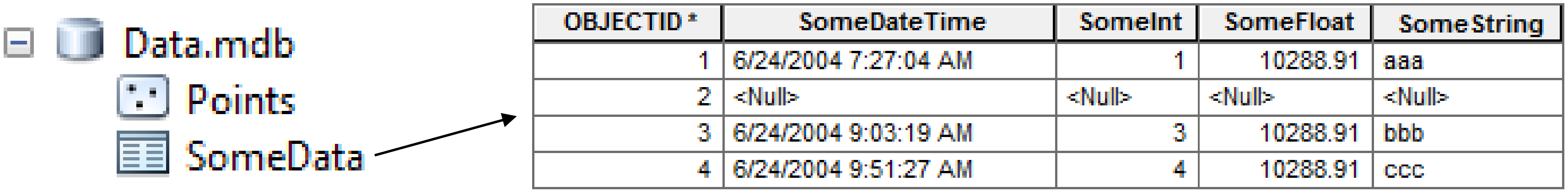
With the arcgisbinding package:

Initialize the ArcGIS license:

```
> library(arcgisbinding)
*** Please call arc.check_product() to  define a desktop license.
>
> arc.check_product()
product: ArcGIS Desktop
license: Advanced
build number: 10.3.1.4959
binding dll: rarcproxy
>
```

With the arcgisbinding package:

Read a table into R:



OBJECTID *	SomeDateTime	SomeInt	SomeFloat	SomeString
1	6/24/2004 7:27:04 AM	1	10288.91	aaa
2	<Null>	<Null>	<Null>	<Null>
3	6/24/2004 9:03:19 AM	3	10288.91	bbb
4	6/24/2004 9:51:27 AM	4	10288.91	ccc

```
> dataset <- arc.open("D:/Temp/Data.mdb/SomeData")   # Open the dataset
> arcdf <- arc.select(dataset)    # Get an arc.data instance of data.frame
> summary(arcdf)
 OBJECTID     SomeDateTime        SomeInt             SomeFloat        SomeString
 Min.   :  1   Min.   :38162   Min.   :-2.147e+09   Min.   : 8395   Length:949
 1st Qu.:238   1st Qu.:38171   1st Qu.: 2.380e+02   1st Qu.: 9862   Class :character
 Median :475   Median :38180   Median : 4.750e+02   Median :10011   Mode  :character
 Mean   :475   Mean   :38184   Mean   :-2.262e+06   Mean   :10009
 3rd Qu.:712   3rd Qu.:38194   3rd Qu.: 7.120e+02   3rd Qu.:10155
 Max.   :949   Max.   :38211   Max.   : 9.490e+02   Max.   :11274
               NA's   :1                            NA's   :1
```

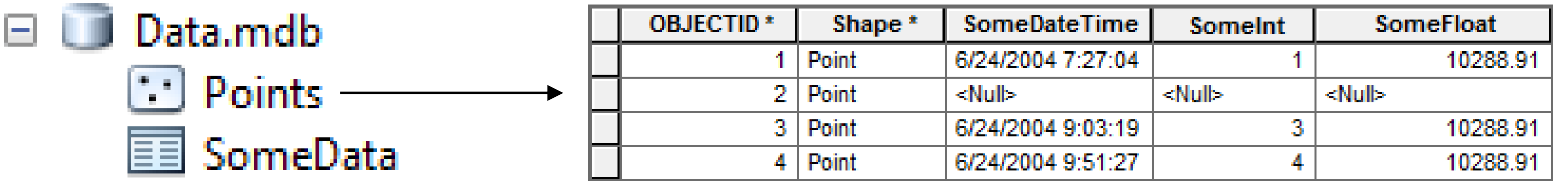
Strings not automatically converted to factors (good, in my opinion)

NULL integers converted to -2147483647

Datetime values converted to floating point (number of days since 1899-12-30?)

With the arcgisbinding package:

Read a feature class into R:



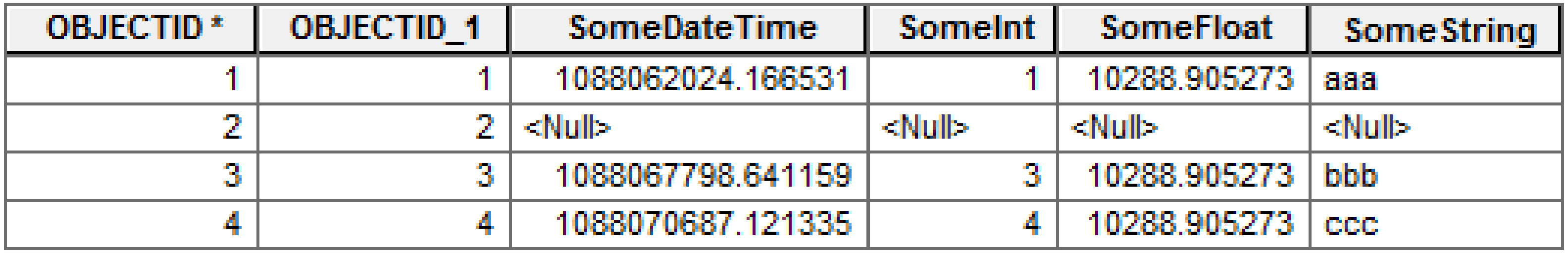
	OBJECTID *	Shape *	SomeDateTime	SomeInt	SomeFloat
	1	Point	6/24/2004 7:27:04	1	10288.91
	2	Point	<Null>	<Null>	<Null>
	3	Point	6/24/2004 9:03:19	3	10288.91
	4	Point	6/24/2004 9:51:27	4	10288.91

```
> dataset <- arc.open("D:/Temp/Data.mdb/Points")   # Open the dataset
> arcdf <- arc.select(dataset)       # Get an arc.data instance of data.frame
> points <- arc.data2sp(arcdf)       # Convert to SpatialPointsDataFrame object
> library(sp)                        # Necessary to access sp functions
> summary(points)
Object of class SpatialPointsDataFrame
Coordinates:
                min      max
coords.x1 -703555.8 633107.0
coords.x2 -663940.9 793006.7
Is projected: TRUE
proj4string :
[+proj=aea +lat_1=38 +lat_2=30 +lat_0=34 +lon_0=-73 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0]
Number of points: 949
Data attributes:
    OBJECTID     SomeDateTime        SomeInt             SomeFloat       SomeString
 Min.   :  1   Min.   :38162   Min.   :-2.147e+09   Min.   : 8395   Length:949
 1st Qu.:238   1st Qu.:38171   1st Qu.: 2.380e+02   1st Qu.: 9862   Class :character
 Median :475   Median :38180   Median : 4.750e+02   Median :10011   Mode  :character
 Mean   :475   Mean   :38184   Mean   :-2.262e+06   Mean   :10009
 3rd Qu.:712   3rd Qu.:38194   3rd Qu.: 7.120e+02   3rd Qu.:10155
 Max.   :949   Max.   :38211   Max.   : 9.490e+02   Max.   :11274
               NA's   :1                            NA's   :1
```

With the arcgisbinding package:

Write a table or feature class from R:

```
> summary(df)
    OBJECTID    SomeDateTime                    SomeInt         SomeFloat      SomeString
 Min.   :  1   Min.   :2004-06-24 07:27:04   Min.   :  1.0   Min.   : 8395   Length:949
 1st Qu.:238   1st Qu.:2004-07-03 11:28:26   1st Qu.:238.8   1st Qu.: 9862   Class :character
 Median :475   Median :2004-07-11 13:18:43   Median :475.5   Median :10011   Mode  :character
 Mean   :475   Mean   :2004-07-15 14:40:21   Mean   :475.5   Mean   :10009
 3rd Qu.:712   3rd Qu.:2004-07-26 11:06:12   3rd Qu.:712.2   3rd Qu.:10155
 Max.   :949   Max.   :2004-08-11 18:58:50   Max.   :949.0   Max.   :11274
               NA's   :1                     NA's   :1       NA's   :1

> arc.write("D:/Temp/Data.mdb/SomeData2", df)
```



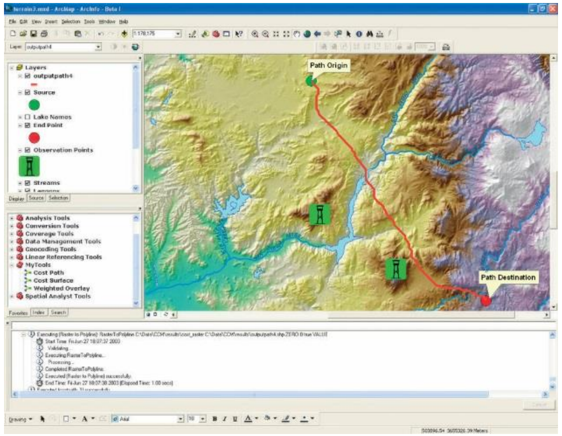
OBJECTID *	OBJECTID_1	SomeDateTime	SomeInt	SomeFloat	SomeString
1	1	1088062024.166531	1	10288.905273	aaa
2	2	<Null>	<Null>	<Null>	<Null>
3	3	1088067798.641159	3	10288.905273	bbb
4	4	1088070687.121335	4	10288.905273	ccc

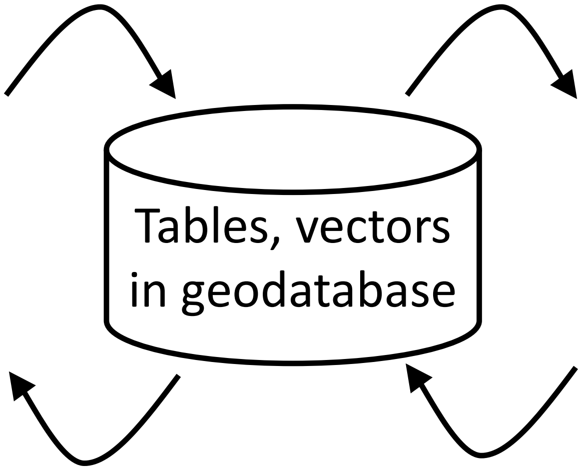
Assigned new OBJECTID,
renamed our column

Converted POSIXct values to floating point
(number of seconds since 1970-01-01?)

Recommended approach

Read:
- For tables in personal GDB, use RODBC
- Otherwise use rgdal or arcgisbinding

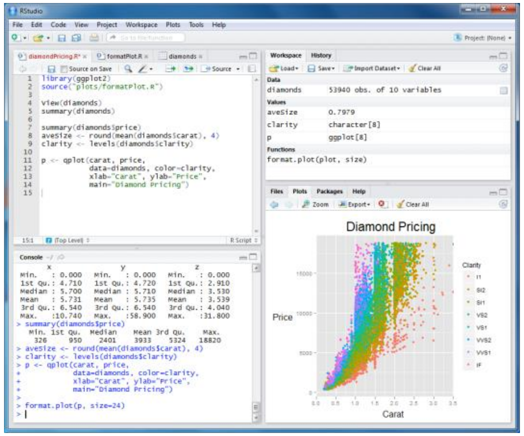




Write

Tables, vectors in geodatabase

Read



```r
library(ggplot2)
source("plots/formatPlot.R")

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
      x               y               z
 Min.   : 0.000  Min.   : 0.000  Min.   : 0.000
 1st Qu.: 4.710  1st Qu.: 4.720  1st Qu.: 2.910
 Median : 5.700  Median : 5.710  Median : 3.530
 Mean   : 5.731  Mean   : 5.735  Mean   : 3.539
 3rd Qu.: 6.540  3rd Qu.: 6.540  3rd Qu.: 4.040
 Max.   :10.740  Max.   :58.900  Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

Data
diamonds 53940 obs. of 10 variables

Values
aveSize 0.7979
clarity character[8]
p ggplot[8]

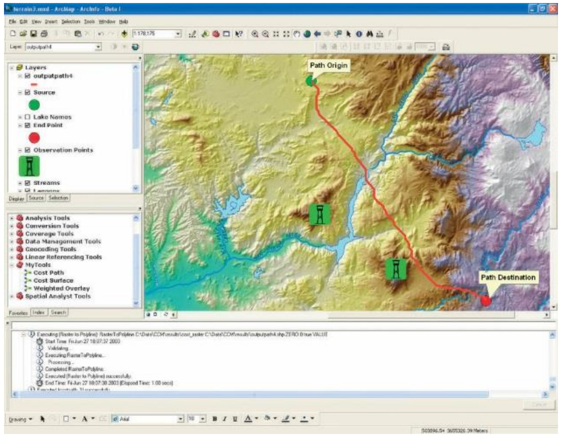
Functions
format.plot(plot, size)

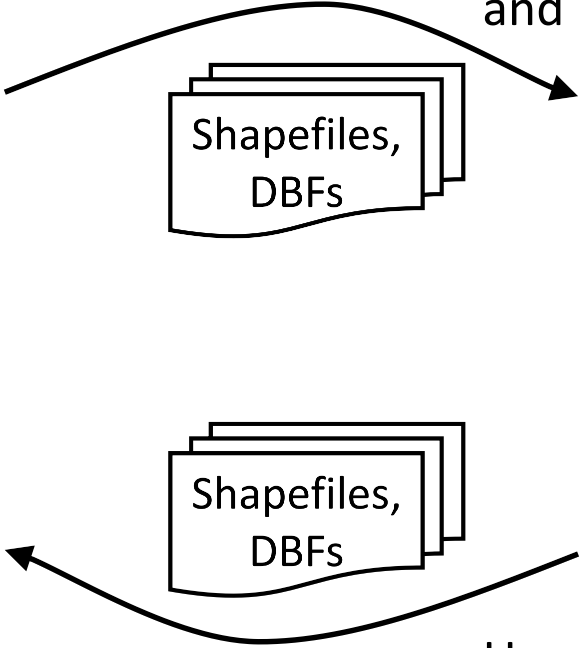
Write:
- For tables in personal GDB, use RODBC
- Otherwise use arcgisbinding

Alternative approach:

If you can tolerate the limitations of shapefile and DBF

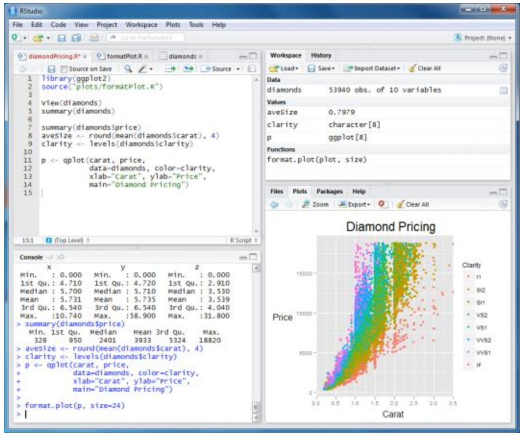
Use readOGR (from rgdal)
and read.dbf (from foreign)





Shapefiles,
DBFs

Shapefiles,
DBFs



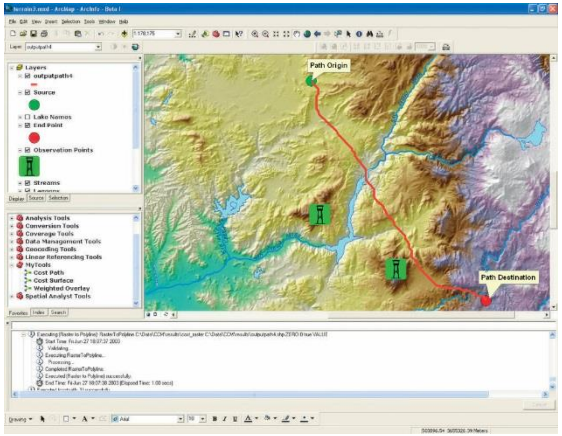
```
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

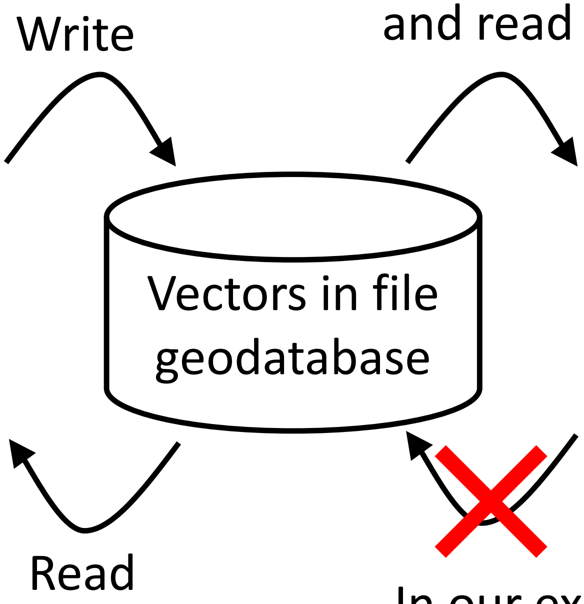
```
       x                y                z
Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median : 5.700   Median : 5.710   Median : 3.530
Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

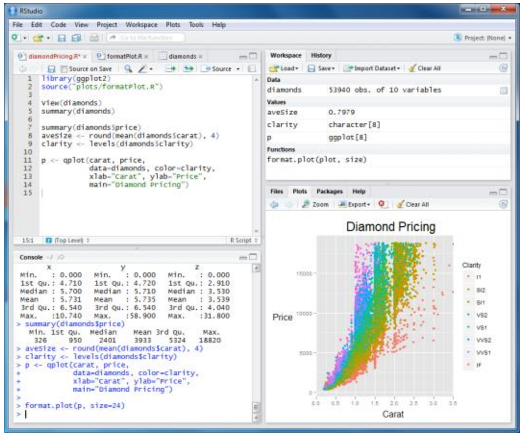
Use writeOGR (from rgdal)
and write.dbf (from foreign)

In this workshop

We only need to send vector data from ArcGIS to R. We will use a file GDB to facilitate cross-platform use and read from it with rgdal.







```
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
Data
diamonds      53940 obs. of 10 variables
Values
aveSize       0.7979
clarity       character[8]
p             ggplot[8]
Functions
format.plot(plot, size)
```

```
       x                y                z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

In our exercise, we do not need to send tables or vector data from R back to ArcGIS.

Rasters

Reading a raster into R:

```
> library(raster)
> r <- raster("D:/Temp/Depth.img")
> r
class       : RasterLayer 
dimensions  : 1260, 1200, 1512000  (nrow, ncol, ncell)
resolution  : 0.01666667, 0.01666667  (x, y)
extent      : -82, -62, 24, 45  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0 
data source : D:\Temp\Depth.img 
names       : Depth 
values      : 0, 6282  (min, max)
```

Writing a raster from R:

```
> writeRaster(r, "D:/Temp/Depth2.img")   # Options for data type, compression, etc.
```

For raster data, I recommend .IMG format

- Supports all pixel types, raster attribute tables, statistics, compression, and very large dimensions
- GeoTIFF is an acceptable alternative, but less flexible, in my experience

Let’s read our sightings into R…