Ten commandments for good data management

https://dynamicecology.wordpress.com/2016/08/22/ten-commandments-for-good-data-management/
Posted on August 22, 2016 by Brian McGill

Usually when I am asked to give a few words to describe myself I say macroecologist or large-scale-ecologist. And I might on other days say biodiversity scientist or global change scientist. But a lot of days I would say "ecoinformatician". Ecoinformatics is the subset of bioinformatics that applies to ecology – that is to say informatic (data) techniques applied to ecology. Some of you may know that I spent 9 years in business before returning to my PhD. But not many know that most of what I was doing was business informatics. Helping companies understand their data. It wasn't planned. I just have always liked seeing what the data has to tell me. But it turned out to be great training as ecology dived into informatics just as I hit graduate school.

Not surprisingly given my background, I spend a lot of time being asked to make recommendations on how to work with data. I've also been involved in some very large data projects like BIEN. Here I don't want to focus on the large (often social) issues of really big projects (my slides from ESA 2015 on the next 100 years of ecoinformatics are on figshare if you're interested). Here I want to focus on the much smaller single person or lab-scale project. This post is attempts to summarize what I have learned to be best practices over both my business informatics and ecoinformatics careers. I am intentionally going to stay tool or software agnostic. In this post I really want to emphasize a frame of mind and mental approach that can be implemented in literally dozens of different software packages. In a second post tomorrow, I will give a worked example in R since I know that has the highest popularity in ecology.

Warning – this is a really long post! But I hope it is useful.

I want to emphasize I don't assume I have the only truth. There are plenty of other knowledgeable people out there (and I hope they chime in). And I'm sure some of them would disagree with me on at least some points. So definitely take what you like and leave the rest. And if this stokes your interest, definitely check out Data Carpentry about arranging classes (but they take no responsibility for what I say in this post as I think they are great but I am not affiliated with them).

So my 10 commandments for managing research data are:

1 – Thou shalt distinguish raw data from analytical data and link them with a repeatable pipeline

One big hurdle I see people stumble on over and over is conceiving of there being one copy of The Data. This may be how the human brain works. But its not how computers work best. Computers are stupid and highly inflexible (albeit very fast). Data needs to be structured for what you want the computer to do with it. I would suggest that at a minimum you should conceive of two types of data (and create both types for a single project):

1. **Raw data** – this is where you do data entry and collect and assemble your data. It will be at the level detail of your measurements and lab notebooks. And it will be structured to minimize data entry errors and promote error checking.
2. **Analysis data** – this is what you feed into various visualization, statistics and analysis routines. This data may be summarized from the raw data (e.g. summarized to weekly or monthly periods). It will almost certainly be organized into a different structure to facilitate analysis rather than data entry. You may even have multiple levels of summary for different analyses (making multiple analysis data sets).

I have seen many people start with raw data and then gradually over time morph it into analysis data without realizing that was what they were doing and in the process destroying the raw data. Much better to have a clean break. The raw data comes straight out of your lab notebooks (or is entered directly into a computer in the field) and then is never modified again. The analysis data is derived through a repeatable pipeline (aka computer scripts that can be run over and over again) that can take the raw data as input and spit out the analysis data as output. This gives you data optimized for the task (entry or analysis) but also gives you essentially one data set from a reproduceability point of view since one button click will regenerate the analysis data.

2 – Thou shalt create raw data in instance-row, variable-column format

Many people like to capture their raw data in crosstab form (Figure 1 – also called spreadsheet style and sometimes table form, but the term "table" is ambiguous so for reasons that will become apparent I am going to call this the dimensional view). But putting raw data in a dimensional view is really confusing raw data and analysis data goals.

	2010	2011	2012
Washington	20	22	18
Oregon	10	13	5

Figure 1 – Dimensional View

I would argue that raw data should ALWAYS be captured in the format where each measurement has one row, and each column is a different variable or attribute (Figure 2 – also confusingly called table view or database or dataframe style) so here called instance-row, variable-column or just row-column format. Variables might be e.g. latitude or longitude where measured or a site code, or a species name or individual code or day collected or abundance or body mass or …

Site	Year	Abundanc
Washington	2010	20
Washington	2011	22
Washington	2012	18
Oregon	2010	10
Oregon	2011	13
Oregon	2012	5

Figure 2 – row-column view

More commandments from Roel (Roel.klink@idiv.de)

- **NEVER EVER put more than one piece of information in a cell, this may be fast for data entry but is incredibly hard to separate later. The worst is when the pieces of information have inconsistent number of characters or no delimiters. Example: 'SitenamePlotnameDate' or 'GenusSpeciesSex'**
- **Never use multiple column headers**
  example: site 1 site 1 site1 site 2 site2 site2
  date1 date2 date3 date1 date2 date3
- **Don't repeat information in rows or columns that can better be calculated e.g.: males, females, total, or: total, females**

Note that in this case the row-column view and the dimensional view are just different ways of viewing the same data. That is kind of the point of commandment #1. The dimensional view is much more intuitive to analyze (e.g. it quickly makes obvious that Washington has higher abundance than Oregon). It also makes the dimensions obvious (here space and time) as each dimension is one "direction" in the table (horizontal or vertical). But it has many problems for capturing data. First the metadata structure is more obvious in the row-column format (where do the year & site & abundance titles go in the dimensional view). And what if we add species as a 3rd dimension – how do you make that table? And what if we didn't collect in Oregon 2011? This is obvious as an absent row in the row-column view but how do we show it in the dimensional view? as a 0? an NA? an *? This highlights the power of the row-column view in that every row has an exact 1-1 mapping with a data point. This is not necessarily true in the dimensional view.

Note that there are lots of good places to store row-column format data. Simple CSV (comma separated variable) files, tab-delimited files, R data frames, a SQL table in Postgres or MySQL are all good choices but you can do it in Excel too – its more a mindset than a technology issue.

3 -Thou shalt partially (but no more and no less) normalize raw data into star schema(s)

Data scientists love to talk about normalization. And ecologists are blown away by this concept and do whatever the data scientists say (fully normalize the data). But this is wrong. Most ecology datasets should only be partly normalized.

What is normalization? The concept is easy and might be better phrased in English as factoring. Imagine you are collecting data on individual birds living in lakes (such as sex and body size). You also have some data that only makes sense at the species level (e.g. total abundance of birds observed in the Breeding Bird Survey). And some that is only available at the family level (e.g. diet). You might have a hypotheses tying together sexual dimorphism with abundance and diet. You could store this data in several equivalent ways. Look at the following three figures.

Family	Diet	SpeciesName	Abundance	IndivID	Weight_g	Sex
Anatidae	Plant	Wood Duck	508.2	LJ001	650	M
Anatidae	Plant	Mallard	5747.2	LJ002	1050	M
Gaviidae	Fish	Pacific Loon	10	LJ003	4126	M
Gaviidae	Fish	Common Loor	277.4	LJ004	4002	F
Anatidae	Plant	Wood Duck	508.2	PD001	605	F
Anatidae	Plant	Mallard	5747.2	PD004	1098	M
Anatidae	Plant	Wood Duck	508.2	PD006	623	M
Anatidae	Plant	Mallard	5747.2	WN001	1058	M
Gaviidae	Fish	Common Loor	277.4	WN005	4400	F

Figure 3 – Fully denormalized

Family	Diet
Anatidae	Plant
Gaviidae	Fish

Family	SpeciesName	Abundance
Anatidae	Wood Duck	508.2
Anatidae	Mallard	5747.2
Gaviidae	Pacific Loon	10
Gaviidae	Common Loon	277.4

SpeciesName	IndivID	Weight_g	Sex
Wood Duck	LJ001	650	M
Mallard	LJ002	1050	M
Pacific Loon	LJ003	4126	M
Common Loon	LJ004	4002	F
Wood Duck	PD001	605	F
Mallard	PD004	1098	M
Wood Duck	PD006	623	M
Mallard	WN001	1058	M
Common Loon	WN005	4400	F

Figure 4 – Almost completely normalized

Family	Diet	SpeciesName	Abundance
Anatidae	Plant	Wood Duck	508.2
Anatidae	Plant	Mallard	5747.2
Gaviidae	Fish	Pacific Loon	10
Gaviidae	Fish	Common Loor	277.4

SpeciesName	IndivID	Weight_g	Sex
Wood Duck	LJ001	650	M
Mallard	LJ002	1050	M
Pacific Loon	LJ003	4126	M
Common Loon	LJ004	4002	F
Wood Duck	PD001	605	F
Mallard	PD004	1098	M
Wood Duck	PD006	623	M
Mallard	WN001	1058	M
Common Loon	WN005	4400	F

Figure 5 – partly normalized

At one extreme we have the top version (Figure 3) which is fully denormalized (i.e. all smashed into one table with lots of redundant information). At the other extreme we have the middle version (Figure 4) where the data is nearly fully "normalized" or "factored"* and all redundancies are eliminated by creating separate tables (namely family, species, and observation tables). Figure 5 is a halfway house and is my recommended approach.

The main advantage of normalization is that it avoids redundant information. Figure 3 repeats the information that a Wood Duck is in the family Anatidae and eats plants 3 times (and also repeats that the average abundance of a Wood Duck is 508.2 three times). Figure 4 only repeats each of these facts once. Note how Figure 4 has a different kind of redundancy – the column Family appears in two tables so the two tables can be linked together (as does species name). From a data entry point of view Figure 4 has a lot to recommend. Often cited is that it is typically smaller. This is a worthless reason – disk space is cheap and most ecological

datasets are quite small. But a much more valid argument is that fewer redundancies are fewer opportunities for errors in data entry to creep in. But in my experience computers may be great at working with denormalized data but humans get lost quickly as the number of tables grow. So, I find that as is so often the case, compromise is the best option. So I favor partial denormalized data (Figure 5, the last example).

In fact I favor I very specific schema (that is the database techie word for the design of data tables to use) that have been around in the business world for decades known as a "star schema", which is partially de-normalized. A star schema has one central row-column table known as the "fact" table. In this table each row corresponds to measurement on one entity. And the columns of the "fact table" are either dimensions (to be defined in a minute) or measurements (like body mass and sex). The right-most table in Figure 5 (or Figure 4) is a fact table. The "dimensions" are the "coordinates" of the facts. In ecology space, time and taxa are extremely common dimensions. Figures 3-5 have only one dimension (taxa). Figures 1 and 2 have two dimensions (space, time).

A star schema therefore has one fact table and one table for each dimension. The fact tables are linked to the dimension tables by unique identifiers like species name, date, site code, or site lat/lon (i.e. these columns are duplicated and found in both the fact table and one dimension table). Note that a dimension table will have many fewer rows than the fact table. Indeed the space dimension table will have one entry for each site that you observed at. The taxa dimension will have one entry for each taxa (or individual) that you observed. Thus if you have say 20 sites, 100 species, and 3 years you will have one space dimension table with 20 rows, one taxa dimension table with 100 rows, and one time dimension table with 3 rows. Your fact table could have as many as 20*100*3=6,000 rows (if every species is found at every site every year), but in all likelihood will have somewhat fewer rows. To me this is the "right" amount of denormalization or factoring. Basic attributes of a species are "normalized" and pulled out in the taxa dimension table and not repeated for every observation. But families and species are denormalized (aka repeated) in the fairly small dimension tables leading to a little bit of redundancy. Again this is called a "star schema". An example is in Figure 6.

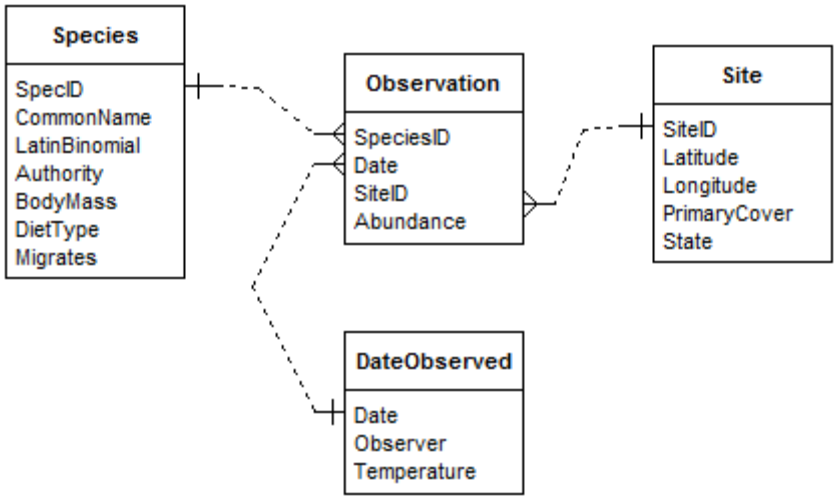

Figure 6 – Star schema example

In this figure each rectangle is one row-column table. The column names are listed in the rectangle. In this example "Observation" is the fact table. It stores one measurement,

abundance, and three dimensions (taxa/Species, time/DateObserved, and space/Site). Note that the Site dimension table is linked to the fact table by the shared column "SiteID". This is denoted in this type of drawing (again known as a database schema) by the line linking SiteID in the two tables. The "crowsfeet" shows the direction of the many-to-one relation (i.e. there are many entries in the Observation table for one entry in the Site table). If you want you can use software known as Entity-Relationship or ER software to draw these diagrams. I used the free program ER Assistant to draw this. But that is probably overkill for most people – pen and paper is more efficient unless you do this a lot.

The right way to use a star schema in data entry is the followin:

1. Figure out the dimensions of your data (space, time and taxa are usual suspects but you may have more or less)
2. Create your dimension tables (the small tables that surround your fact table) first. Some people would call this a data dictionary or in social sciences "the codebook".
3. Enter your data into the fact table
4. Either in real time as you enter the data, or periodically, check your entries in the fact table to make sure they actually point to entries in the dimension tables (this can be done by join operations which are discussed below).

The origins of the name "star schema" may be obvious in Figure 6 – it has one big central table and lots of little tables coming off it. It looks (if you squint) like a star. And in case you are wondering, you may have more than one fact table. A common example is where you have some organismal measurements (abundance, body size) with space, time and taxa dimensions. And then you have environmental measurements (temperature, soil pH) with space and time dimensions. It might not surprise you given how cute computer scientists think they are that a star schema with multiple fact tables is called a "constellation schema". My post tomorrow giving an example in R on this topic will use a constellation schema (one biotic fact table, one environmental fact table). If you're into cute names, if you take a star schema but then fully factor/normalize the dimension tables into multiple tables (go from Figure 5 to Figure 4) it is called a snowflake schema (you have to really squint to see a snowflake) (But don't let computer scientists talk you into using snowflake schemas!).

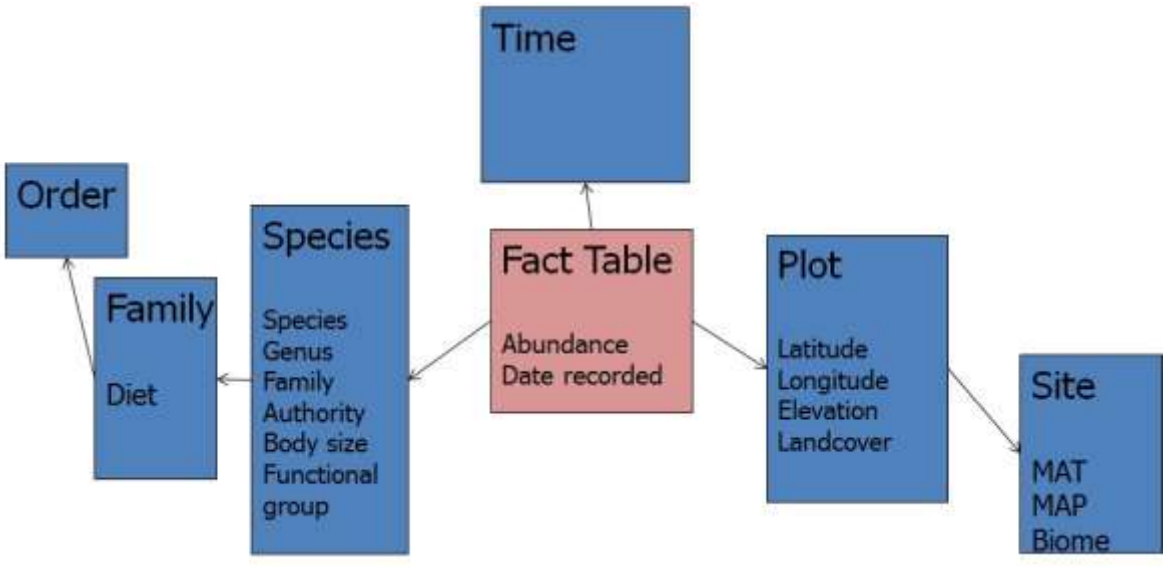

Figure 7 – Snowflake schema with 3 dimensions (space, time, taxa) but the dimension tables are normalized

Note that again a star schema can be implemented anywhere that can implement row-column data. For example, each fact or dimension table can be a separate CSV file. Or they can be separate tables in a SQL database. Or separate tabs in an Excel spreadsheet. Again, its a mindset, not a technology.

4 – Thou shalt plan ahead to spend a lot of time cleaning your data

Over twenty years ago, Gartner group (a consulting company that advises business on how to keep up with computer technology) said that 70% of the work in building a data warehouse was cleaning the data. Now datawarehouse is the business buzz word for what I called analysis data above. Businesses capture raw data in transactional systems like cash register point of sale systems, accounting software, etc. This then gets transformed (by a reproducible pipeline) into what business calls a datawarehouse and what I call analysis data. Gartner's point is you might think if you already have the data in one form it should be quick to go to the other. But Gartner estimated that 70% of the work (i.e. a huge amount of work) goes into cleaning up that transactional data. Store A gives product X one barcode, while Store B gives the same product X a different barcode. And these have to be reconciled. Or field assistant A calls a species by its old name, and field assistant B uses the newest taxonomic revision. Or field assistant A enters states using their two letter postal codes (Oregon is OR), but field assistant B is from Europe and makes up 2 letter codes, or is American and forgets whether MI is Michigan or Mississippi or spells them all out. But in the end, long story short, expect to spend at least half your project from data entry to anlaysis cleaning up your data. Plan for it. Do it up front. Don't just fix things when they give you funny results because then you won't have fixed a lot of things that give you non-obviously wrong results.

As an aside, note that one of the benefits of treating raw data separate from analysis data is you can do somethings to minimize these problems. As discussed above, you can easily write code that checks that every entry (row) in the fact table matches out to an entry in each dimension table (e.g. if I enter a fact with state as "Miss" is there a row in the space dimension table that has a state of "Miss" (probably not since this is a non-standard abbreviation). If you're really fancy you can do these checks real time as somebody enters the data. Note that this assumes that you create your dimension tables first, then your fact table.

5 – Thou shalt not hand-edit raw data

Once entered – you should never change your raw data again. All those errors you find in step #4 (e.g. entering "Miss" instead of "MS" for the state of Mississippi) should be modified and corrected in a new "corrected raw data" copy of the data populated by scripts. It might seem clunky to go find all the entries of MI in state for field assistant B and change them to MS instead of just go edit them by hand. But you will regret it if you edit by hand in the long run. Two main benefits of putting it in a script: 1) All those times you thought you found an error and "fix" them only to discover you misunderstood and introduced an error are safely covered because you still have the raw data. 2) You have documented everything you have done and the corrections are reproducible (and reusable if a new year of raw data comes in). This inherently requires a programming language, but there are still many choices. There are tons of tools for data cleaning. In business they go under the generic name of ETL (extract, transform and load). Data Carpentry uses OpenRefine. I like awk. Many bioinformaticians prefer PERL. You could even write VBA scripts in Excel. And tomorrow I will give some examples in R. But hand editing Excel spreadsheets is right out. It has to be a script you can run over and over.

6 – Thou shalt conceptualize analyses as dimensional

What shape is your data? This may seem like an odd question (most would answer "rectangular" if they had any answer at all). But when you start thinking dimensionally it becomes a more relevant question. When thinking about analyzing data, I find it very helpful to think about it as a multidimensional cube:

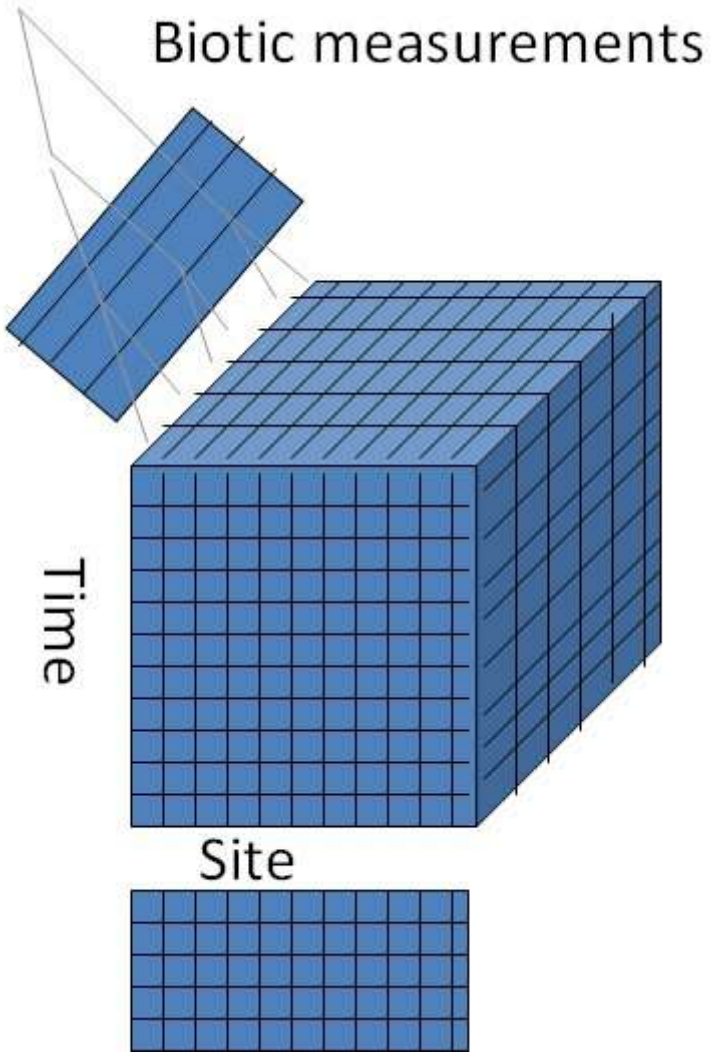

Figure 8 – Multidimensional view of a star schema

In Figure 8 – the fact table is visualized as a cube with three dimensions of time, space/site and species. Each "cell" in this cube holds an abundance for the particular time/site/species combination it is at the intersection of. This makes the notion of dimensions as the "coordinates" of the data very explicit. And the dimension tables are shown as ancillary tables (time doesn't have one here and species has a phylogeny overlayed as well).

Remember when I explained why the dimensional (crosstab/spreadhseet) view of data had problems when you go into three dimensions in Commandment #2? Well it does have problems when you try to represent in say Excel. But you can visualize and conceptualize it really well. And in fact this is how I conceptualize every dataset I work with. It then becomes really obvious to start thinking how to summarize data. Perhaps you want to average across time, and sum across sites, leaving a vector (1-dimensional list) of species (which can still be

easily aligned with the taxa dimension table). And so on. You can still do all of these operations on a row-column table (and per commandment #7 below you may end up doing exactly this). But I find it easier to think about data dimensionally. Of course I might be biased since I spent several years as the product manager of a multidimensional database named Acumate (now defunct but a detailed description of multidimensional databases can be found here albeit in a business context). For a while there was also a market for multidmensional spreadsheets.

Just like you can have multiple fact tables you can have different multidimensional cubes (and they may not all have the same number of dimensions). This is a good way to think about ways in which your different fact tables overlap and don't overlap. Here is a two fact table/cube representation of the constellation schema I showed before (Figure 7).

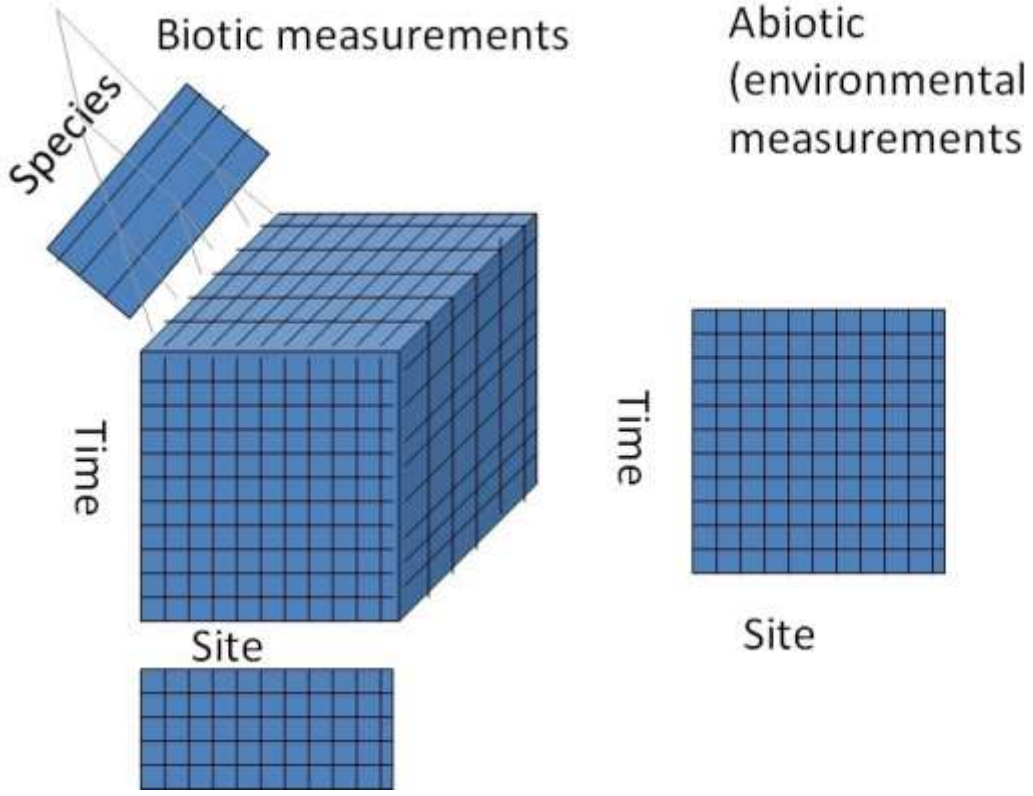

Figure 9 – Two cubes

7 – Thou shalt store analytical data as denormalized but optimal for the tool you are using

Analysis should ALWAYS be done on a fully denormalized data (i.e.like figure 3) where everything is mashed into one big table and redundant. Your star schema that you used for data entry should be collapsed back into a single data table for analysis. You do this by a process known in the database world as a "join". A join is simply combining two tables by a field they have in common (e.g. SpeciesName in the example above). The join is a horizontal concatenation – i.e the tables are joined side by side or to put it in other terms – every column in table 1 and every column in table 2 become columns in the joined table (except the joining column which is not duplicated but shows up only once in the joined column). It is basically going from Figure 5 to Figure 3 (i.e. a a join is the opposite of a normalization or factorization). In a star schema you will still have the same number of rows as your fact table,

you will just have a lot more columns in your single denormalized table. In theory one can do joins on the fly depending on the analysis you want. But I say why bother. Once you have your data cleaned up in a star schema, do one last step and join it all together into one big table.

You can do joins in any software. You can use the merge command in R. Or you will see in the next post I recommend using dplyr and join commands in R. You can do joins in SQL (the syntax is beyond this post but found in any introduction to SQL). You can do joins in Excel using lookup tables. Its a core concept in data management.

How should you store your one big table for analysis? Some software directly supports the multidimensional view. For example n-D arrays in Matlab or NumPy arrays in Python. In theory you could do it with n-D matrices in R but R doesn't really like n-D matrices and one of the problems with cubes is they explode very quickly (100 species, 100 sites, 100 times gives 1,000,000 cells) and R's poor memory management bogs down quickly. In other software you are much better to keep the data actually in row-column form but just think of it multidimensionally. Dataframes in R or Pandas data in Python are good examples of this. I actually use a mix of row-column and multidimensional cube storage in Matlab when I am coding. Usually I start with row-column when the data is really big and after I've filtered it I convert it into a muldimensional cube, then summarize it and run statistics on it. The point is you can have a conceptual model that is multidimensional and a data model that makes your analysis software happy.

You can do dimensional-like summarization in any software. They're called pivot tables in Excel. They are the apply family of commands or the by command in R combined with commands like sum and mean. Or in dplyr package in R they are group_by and summarise commands. These have direct analogies to the group by and summary commands in SQL.

8 – Thou shalt create analytical data and analyze it by reproducible code

If there has been one consistent theme, its don't ever do anything as a one off. Yes you can create an Excel pivot table from a star schema and summarize it using some points and clicks on menus and dialogue boxes and save the results out. But can you reproduce it if you get new data? Everything to get to the fully denormalized analysis table should be done by scripts that can be run repeatedly and reproducibly and reusably.

9 – Though shalt obsessively hand check the transformation from raw data to analytical data

So you may have noticed that I recommend a number of steps to get from raw data entry to analysis. Thou shalt never assume that simply because the computer code ran and spit out data of the right shape you have the right answer. You almost certainly don't. There are all kinds of ways things can go wrong. It is critically important, indeed, I would say a matter of scientific ethics, that you laboriously hand check your data. Take a row in your analysis table and hand trace it back to the raw data. Can you reproduce that row from the raw data by hand calculations? If so take another entry and trace it back. Do this enough to be really confident it is right. Similarly scrutinize your analysis table. Do you have a species called "NA" (or a time period or site)? How did that sneak in there? Is an endangered species showing up as your most common species? Did your joins work so that a species is lined up with the right family?

Really poke and prod your data. Torture your data to make it confess its errors (a variation of a Ronald Coase quote)

10 – Thou shalt use the simplest feasible technologies to keep the focus on ecology

You will I am sure have noticed that I have repeatedly emphasized that the important thing is the mindset and approach not the technology. You can do everything I suggest here in Excel. You can also do everything I suggest here in an SQL database. Which should you use? Well you have to weigh it out. In particular I think the main trade-off is learning-curve vs scaling. Using more complicated technologies (like an SQL database) is a steep learning curve for almost all ecologists. This weighs heavily towards using familiar tools. But the bigger and more complicated your data is, the more that learning curve starts to pay off in the ease with which these concepts can be put into practice (and the automatic error checking built into more advanced tools). In short, the bigger and more complicated your data, the more it is likely it is worth the learning curve of a newer more sophisticated tool. But don't let a computer savvy person intimidate you into saying you have to do data management in an SQL database. Or that you have to normalize your data.

Summary

So overall I have been arguing for a workflow (or pipeline if you prefer) that looks like (Figure 10):

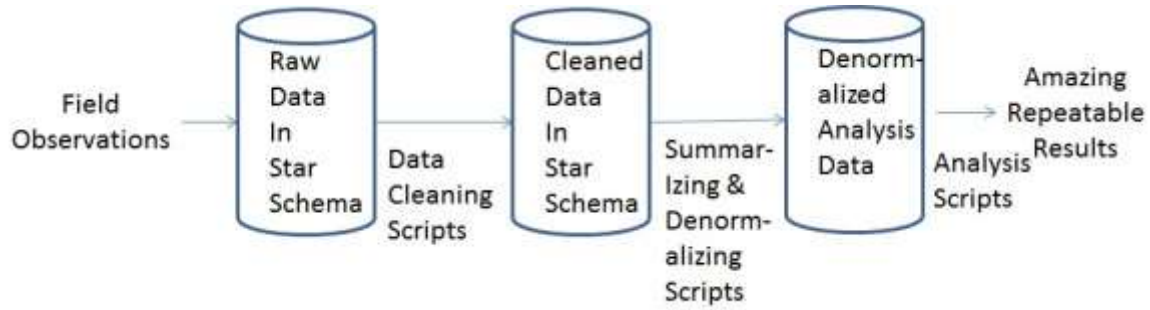

Figure 10 – Pipeline

So you will start with some dimension tables (probably created by hand in advance) and one fact table ("Raw data in Star Schema") that will contain errors. You will clean these errors up by scripts which will create a new star schema that is almost identical in size and shape ("Cleaned Data in Star Schema") and then you will denormalize it and end up with one mega row-column table or multidimensional cube ("Denormalized Analysis Data"). Note that this pipeline calls for 3 "copies" of the data. I put copies in quotes because they are certainly copies (i.e. distinct files that contain redundant information) but if you follow the text on the arrow and create these copies not by hand but by scripts that can be rerun, then in many ways these are really just 3 different views on the data, not 3 copies.

Tomorrow I will post an example dataset and an example R script that moves through the data in agreement with these 10 commandments.

I'm curious. For those of you who think a lot about ecoinformatics, what do you agree or disagree with? And for those of you who are new to these concepts, what have I explained poorly? What needs expansion?

UPDATE – a direct link to the next day's post with sample R code

UPDATE 2 – Eric Lind pointed out this nice paper by Elizabeth Borer et al on data management which matches many of the themes herein.

---

* I say nearly fully normalized because a real hard core normalizer would recognize in the real world we would probably have 60 families and only 5-10 diet types and recommend that diet-type be its own table and family would then have codes pointing into the diet type table. That way there is a master table of diet types where each diet is entered once and spelling scan be checked against it.