# Correlation is not Causation (or, when not to scream “Eureka!”)

## What not to do

Legend has it that Archimedes once solved a problem sitting in his bathtub. Crying *Eureka!* (“I have it!”), Archimedes leapt out of the bath and ran to tell the king about the solution. Legend does not say if he stopped to get dressed first.

When we stumble onto some pattern in the data, it is so tempting to send a *Eureka!* text to the business users. This is a natural response that stems from the excitement of doing science and discovering an effect that no one has ever seen before.

Here’s my warning: don’t do it. At least, don’t do it straight away.

I say this because I have often fallen into the trap of *correlation is not causation*. Which is to say, just because some connection pattern has been observed between variables does not necessarily imply that a real-world causal mechanism has been discovered. In fact, that “pattern” may actually just be an accident- a mere quirk of cosmic randomness.

## Example

For an example of nature tricking us and offering a “pattern” where, in fact, no such pattern exists, consider the following two squares (this example comes from Peter Norvig). One of these was generated by people pretending to be a coin toss while the others were generated by actually tossing a coin, then writing vertical and horizontal marks for heads or tails.

-||--|-|-||-|-||-||-|--|-|  
--||---|--||--|-|--|-|-|--  
---|-|-|--||-|-|||-|--|-||  
--|-|-||--|--||-||-|-|-||-  
-|-||--||-||-||-|-|--|-|||  
|-||||-||-|||-|-|||-||---|  
|-|-|-||--|--|---|-|--||-|  
-|-|||--|-||-||-|-|-||---|  
-|--||----|||-|-||-|-||-|-  
||-|||-|-|-||-|--|-|-||||-  
---||-|-|||--|-|-|---|-|--  
|||--|--|-|-||-||-|-|-||-|  
 (A)  
   
-|-|||-----|-------||--|-  
-||--|||||--|--|-|||-||||  
--||----||-||-|----|--|-|  
||-|-|-|||-||--|||-|-||||  
|-|||-|-|--||-|-|-||--|--  
||-|--|-----|----|---||--  
||---|---|-||||-|||||-|-|  
|---|---||-||||-|-|------  
-|---|-|||-|---||-||-|---  
|||-||----||||||-|||||---  
|-|------||----||-||-----  
-|||-|||-|--|--|-||------  
 (B)

Can you tell which one is really random? Clearly, not (B) since it has too many long runs of horizontal and vertical marks. But hang on– is that true? If we toss a coin 300 times, then at probability 1/4, 1/8, 1/16, 1/32 we will get a run of the same mark that is three, four, five, or six ticks long (respectively). Now 1/32\*300=9 so in (B), we might expect several runs that are at least six ticks long. That is, these “patterns” of long ticks in (B) are actually just random noise.

## Examples from Software Engineering

Sadly, there are many examples in SE of data scientists uncovering “patterns” which, in retrospect, was more “jumping at shadows” than discovering some underlying causal mechanism. For example, Shull et al. reported one study at NASA’s Software Engineering Laboratory that “discovered” a category of software that seemed inherently most bug prone. The problem with that conclusion was that, while certainly true, it missed an important factor. It turns out that that particular sub-system was the one deemed least critical by NASA. Hence, it was standard policy to let newcomers work on that sub-system in order to learn the domain. Since such beginners make more mistakes, then it is hardly surprising that this particular sub-system saw most errors.

For another example, Kocaguneli et al. had to determine which code files were created by a distributed or centralized development process. This, in turn, meant mapping files to their authors, and then situating some author in a particular building in a particular city and country. After weeks of work they “discovered” that a very small number of people seemed to have produced most of the core changes to certain Microsoft products. Note that if this was the reality of work at Microsoft, it would mean that product quality would be most assured by focusing more on this small core group of programmers.

However, that conclusion was completely wrong. Microsoft is a highly optimized organization that takes full advantage of the benefits of auto-generated code. That generation occurs when software binaries are being built and, at Microsoft, that build process is controlled by a small number of skilled engineers. As a result, most of the files appeared to be “owned” by these build engineers even though these files are built from code provided by a very large number of programmers working across the Microsoft organization. Hence, Kocaguneli had to look elsewhere for methods to improve productivity at Microsoft.

## What to do

Much has been written on how to avoid spurious and misleading correlations to lead to bogus “discoveries”. Vic Basili and Steve Easterbrook and colleagues advocate a “top-down” approach to data analysis where the collection process is controlled by research questions, and where those questions are defined *before* data collection.

The advantage of “top-down” is that you never ask data “what have you got?”– a question that can lead to the “discovery” of bogus patterns. Instead, you only ask “have you got X?” where “X” was defined before the data was collected.

In practice, there are many issues with top-down, not the least of which is that in SE data analytics, we are often processing data that was collected for some other purpose than our current investigation. And when we cannot control data collection, we often have to ask the open-ended question“what is there?” rather than the top-down question of “is X there?”.

In practice, it may be best to mix up top-down with some “look around” inquires:

* Normally, before we look at the data, there are questions we think are important and issues we want to explore.
* After contact with the data, we might find that other issues are actually more important and that other questions might be more relevant and answerable.

In defense of a little less top-down analysis, I note that many important accidental discoveries might have been overlooked if researchers restricted themselves to just the questions defined before data collection. Here is a list of discoveries, all made by researchers pursuing other goals:

* North America (by Columbus)
* Penicillin
* Radiation from the big bang;
* Cardiac pacemakers (the first pacemaker was a badly built cardiac monitor);
* X-ray photography;
* Insulin;
* Microwave ovens;
* Velcro;
* Teflon;
* Vulcanized rubber;
* Viagra.

## In Summary: Wait and Reflect Before you Report

My message is *not* that data miners are useless algorithms that torture data till they surrender some spurious conclusion. By asking open-ended “what can you see?” questions, our data miners can find unexpected novel patterns that are actually true and useful– even if those patterns fly in the face of accepted wisdom. For example, Schmidt and Lipson’s Eureqa machine can learn models that make no sense (with respect to current theories of biology) yet can make accurate predictions on complex phenomena (e.g.  
ion exchanges between living cells).

But, while data miners can actually produce useful models, sometimes they make mistakes. So, my advice is:

* Always, always, always, wait a few days.
* Most definitely, **do not** confuse business users with such recent raw results.

In summary. do not rush to report the conclusions that you just uncovered, just this morning. For example, in the case of the Kocaguneli et al. study, if a little more time had been taken reading the raw data, then they would have found the files written by the core group all had funny auto-generated names (e.g. “S0001.h”). This would have been a big clue that something funny was happenning here.

And while you wait, critically and carefully review how you reached that result. See if you can reproduce it using other tools and techniques or, at the very least, implement your analysis a second time using the same tools (just to check if the first result came from some one letter typo in your scripts).

## References

* Victor R. Basili. 1992. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report. University of Maryland at College Park, College Park, MD, USA.
* Easterbrook, Steve; Singer, Janice; Storey, Margaret-Anne; Damian, Daniela; Selecting empirical methods for software engineering research Guide to advanced empirical software engineering 285-311 2008 Springer London
* Ekrem Kocaguneli, Thomas Zimmermann, Christian Bird, Nachiappan Nagappan, and Tim Menzies. 2013. Distributed development considered harmful?. In Proceedings of the 2013 International Conference on Software Engineering (ICSE ’13). IEEE Press, Piscataway, NJ, USA, 882-890.
* Peter Norving, Warning Signs in Experimental Design and Interpretation, http://goo.gl/x0rI2
* Schmidt M., Lipson H. (2009) Distilling Free-Form Natural Laws from Experimental Data, Science, Vol. 324, no. 5923, pp. 81 - 85.
* Forrest Shull, Manoel G. Mendoncaa, Victor Basili, Jeffrey Carver, Jose; C. Maldonado, Sandra Fabbri, Guilherme Horta Travassos, and Maria Cristina Ferreira. 2004. Knowledge-Sharing Issues in Experimental Software Engineering. Empirical Softw. Engg. 9, 1-2 (March 2004), 111-137.