Fuzzy String Matching - a survival skill to tackle unstructured information

"The amount of information available in the internet grows every day" thank you *captain Obvious*! by now even my grandma is aware of that!. Actually, the internet has increasingly become the first address for *data people* to find good and up-to-date data. But this is not and never has been an easy task.
Even if the Semantic Web was pushed very hard in the academic environments and in spite of all efforts driven by the community of internet visionaries, like Sir Tim Berners-Lee, the vast majority of the existing sites don't speak RDF, don't expose their data with microformats and keep giving a hard time to the people trying to programmatically consume their data.

I must admit though, that when I got my hands on angular.js, I was surprise by how MVC javascript frameworks are now trying to "semantify" the HTML via directives... really cool thing with a promising future, but still lacking of standardization.

And of course you see an ever increasing number of web platforms opening up APIs... but we both know, that there is so much interesting data you cannot just query from an API... what do you do? you end up scraping, parsing logs, applying regex, etc.
Let's assume you got access to the information (good for me, that I'm putting this problem aside :))... So we have managed to retrieve semi-structured data from different internet sources. What do you do next?

Connecting the dots without knowing exactly what a dot is

Well, we know, that data is in many cases useful only if it can be combined with other data. But in your retrieved data sets, there's nothing like a **matching key**, so you don't know how to connect sources.
The only thing you have in the two different data sets you are trying to match is item names... they actually look quite similar and a human could do the matching... but there are some nasty differences.
For example, you have a product name called "Apple iPad Air 16 GB 4G silber" on one source and "iPad Air 16GB 4G LTE" on the other hand... You know it is the same product.. but how can you do the matching?

There are several ways of tacking this problem:

- **The manual approach**
  The most obvious one is just manually scanning the fields in both data sources than are supposed to contain the matching keys and creating a mapping table (for example, 2 columns in a csv). It works well when the data sources are relatively small, but good luck with larger ones!.
- **The "I let others do" approach**
  You've certainly heard of crowd sourcing and services like Amazon Mechanical Turk, where you can task a

remote team of people to do manually do the matching for you. It might not be very reliable, but for certain jobs it's a really good option.
- **The "regex" hell** If you have a closer look at your data, you might define regular expressions to extract parts of the potential matching keys (e.g.: gsub('.*?([0-9]+GB).*','\\1', 'Apple iPhone 16GB black') to extract the number of memory GB in the name of a device and trying to match by several fields, not just by one). But there are so many special cases to consider, that you might well end up in a "regex" hell.

Obviously, you'd love to have a better method, or at least a more automatic one to accomplish this task. Let me say it upfront: there's no automatic easy-to-implement 100% reliable approach to that: even those who after reading it started thinking of machine learning approaches can't ever guarantee a 100% reliability because of the nature of the problem (overfitting vs. accuracy). But enough bad news! Let's talk now about the art of the possible:

The Fuzzy String Matching approach

Fuzzy String Matching is basically rephrasing the YES/NO *"Are string A and string B the same?"* as *"How similar are string A and string B?"*... And to compute the degree of similarity (called "distance"), the research community has been consistently suggesting new methods over the last decades. Maybe the first and most popular one was Levenshtein, which is by the way the one that R natively implements in the utils package (adist)

Mark Van der Loo released a package called stringdist with additional popular fuzzy string matching methods, which we are going to use in our example below.

These fuzzy string matching methods don't know anything about your data, but you might do. For example, you see that in a source the matching keys are kept much shorter than in the other one, where further features are included as part of the key. In this case, you want to have an approximate distance between the shorter key and portions of similar number of words of the larger key to decide whether there's a match. This "semantics" usually need to be implemented on top but might well rely on the previously mentioned **stringdist** methods.

Let's have a look at the three variants in R. Basically the process is done in three steps:

- Reading the data from both sources
- Computing the distance matrix between all elements
- Pairing the elements with the minimum distance

The first methods based on the native approximate distance method looks like:

```
# Method 1: using the native R adist source1.devices
```