Data Wrangling With MongoDB

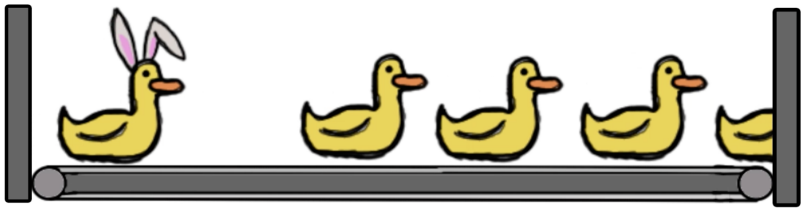
Lesson 03 Notes

Data Quality

Introduction

Now you know how to get data out of a variety of sources. In this lesson we're going to focus on the data itself. How good is it? Can it be trusted? Is it reliable? We're going to look at a blueprint for auditing our data to assess it's quality, and we'll look at a number of different quality measures. We'll also look at ways of cleaning our data to correct any problems that we find with it. As we'll see, this is a process that's very situation specific. But even so, we can identify a number of rules of thumb and best practices that apply in a variety of different contexts.

What is Data Cleaning?



The most important thing to keep in mind about data cleaning, is that it's an iterative process. We iterate on first detecting, and then correcting bad records. Let's look at some examples. We might have text where we expect to find numeric data. So the word two instead of the number two. Some data items might not be designed according to our specification. They might be missing entire fields. They might have extra fields. Or, for example with JSON documents, fields might not have the structure we expect at all. We might have numbers that are out of range. How many people do you know who are minus three years of age? The mighty outliers compared to a standard distribution. We might have an American expatriate in the London office who did some data entry, and coded in a few dates using the U.S. format when the system was expecting the European format.

U.S. 08/24/2013

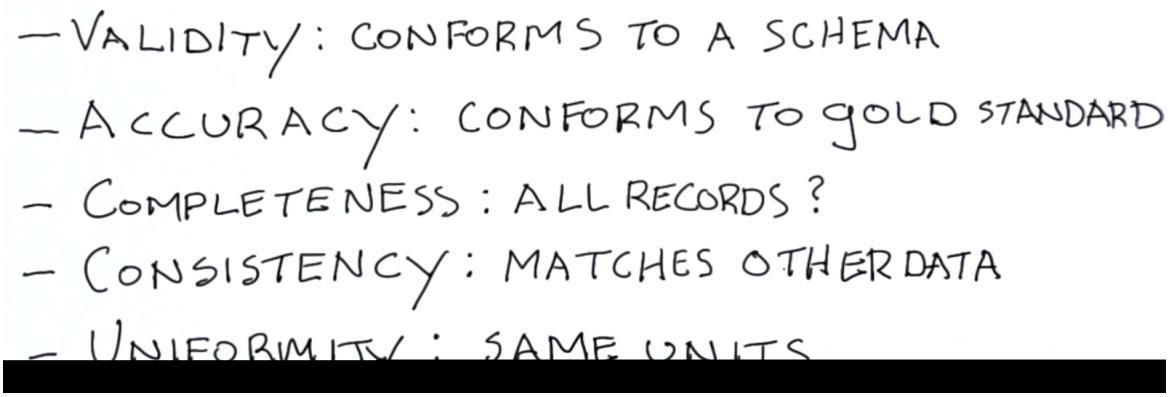
U.K. 24/08/2013

Sources of Dirty Data

There are lots of sources of dirty data. Basically, anytime humans are involved, there's going to be dirty data. This is a lot like any time my kids are involved, there's going to be mud tracked through the kitchen. There are lots of ways in which we touch data we work with. Let me get some hand sanitizer and then we'll get started.
So, we're going to have user entry errors.
In some situations, we won't have any data coding standards, or where we do have standards they'll be poorly applied, causing problems in the resulting data. We might have to integrate data where different schemas have been used for the same type of item. We'll have legacy data systems, where data wasn't coded when disc and memory constraints were much more restrictive than they are now. Over time systems evolve. Needs change, and data changes. Some of our data won't have the unique identifiers it should. Other data will be lost in transformation from one format to another. And then of course there's always programmer error. And finally, data might have been corrupted in transmission or storage by cosmic rays or other physical phenomenon. So hey, one that's not our fault.

Measuring Data Quality

Okay, let's take a look at some formal measures of data quality. There are essentially five measures of data quality that I'd like to talk about. This is the model we'll use in this course. And regardless of who's talking about data quality, and what labels they apply to measures data quality. They're going to fall roughly into these five categories that we'll talk about.

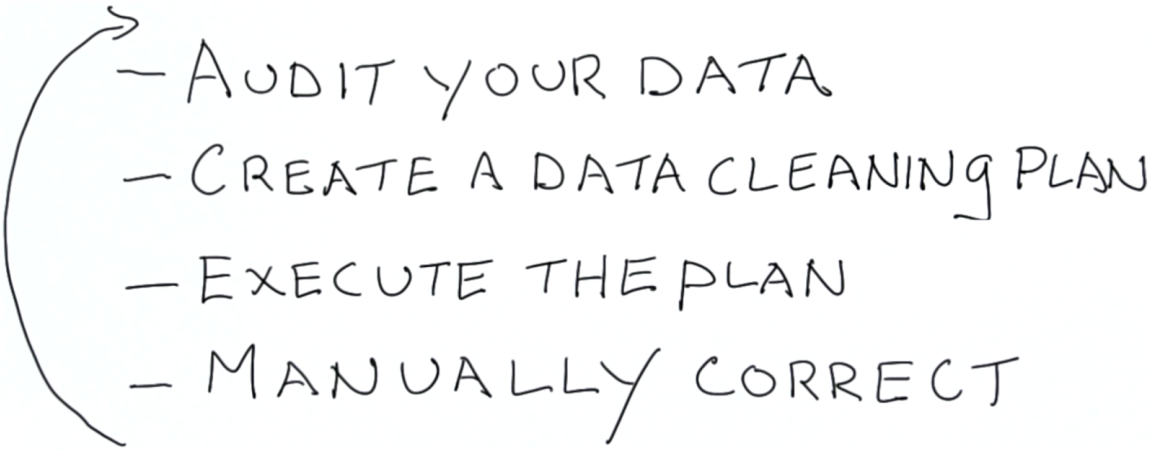


So first let's talk about validity. With validity, we're measuring the degree to which entries in our data set conform to a defined schema, or to other constraints we might have. As we move

through this lesson, we'll discuss each of these in more detail. Here, I'm just going to provide an overview. We can also look at accuracy. This is the degree to which entries conform to gold standard data. What I mean by that, is probably best expressed with an example. So, do all street addresses in a data set that we're cleaning, actually exist? In order to test that type of accuracy question, we would need some gold standard. That is, some set of data that we actually trust. Completeness is pretty straightforward, do we have all the records we should have? While explaining it, is pretty straightforward, actually measuring completeness is a very difficult thing to do. We also want to look for consistency within our data. In many systems, we will have multiple records, that have some overlap in the data they contain. We want to ensure that there is consistency among the fields, that represent the same data across systems. And finally uniformity, this one's easy. Do all our values for distance, for example, use the same units? Is it miles, or is it kilometers?

A Blueprint for Cleaning

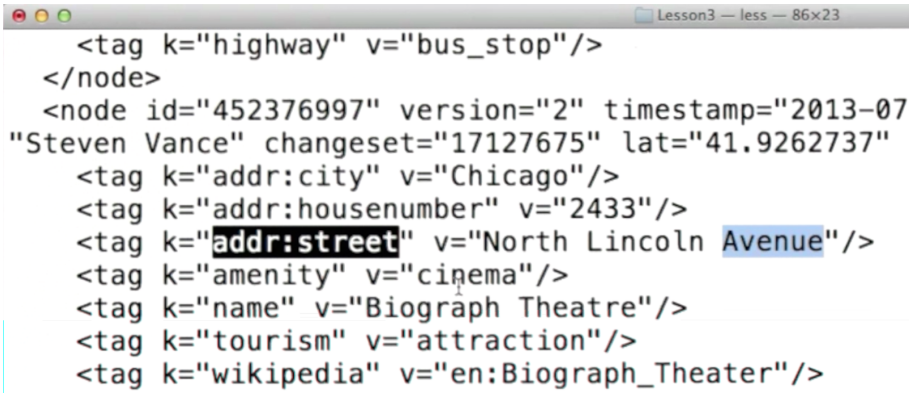
So there's a fairly well established procedure for cleaning data. Now, the specifics are going to very from one situation to another, but the overall process remains the same. Here, we're going to go through this blueprint, or procedure. And then immediately following this discussion, we'll look at a concrete example following this blueprint. So, the first step is to audit your data. Now, in this step what you'll be doing is programatically checking the data using some validation rules you've written, and you'll create a report on the quality of the data. Now, you might also want to run a statistical analysis in order to, for example. check for outliers. Having run your audit, then you use that information to create a data cleaning plan. In our data cleaning plan, we want to identify any causes of the dirty data that we're seeing. Now, this is going to be very situation-specific. Then we're going to define a set of operations that will correct our data, again, situation-specific. And finally what you want to do, is run a few tests to make sure that your data cleaning plan is going to do what you need it to do. Next we're going to execute the plan. Now just to be clear, this is essentially running a script that will execute that set of operations you defined in the data cleaning plan. Now it's possible, in fact it's likely, that we won't have been able to programatically clean our data entirely. And if it's necessary for you to do so, there maybe a manual correction step, where you go through, or you hire someone to go through, and make any changes that need a human eye in order to correct the data.



Now, again this is another process where we have humans involved, so once we've gone through all this, it makes a lot of sense, in my opinion, to iterate on this, possibly more than two iterations, until you have confidence in your data.

Example Using our Blueprint

Okay, let's take a look at an example of applying this procedure we've just talked about. Now in this case, what I want to do is work with the open street map data for the city of Chicago, Illinois in the United States, I chose this because Chicago is my hometown. Now looked at an example from this data set in a previous lesson when we were talking about extracting data from XML. Now, in this example, we're going to take a look at a small part of the cleaning that we would need to do for this data set. This is all human-entered data. Or, for the most part, human-entered data. So, it is especially dirty. What I'd like to look at here is auditing street types. So, for example, avenue. One thing we might be interested in doing with a data set like this is harmonizing all of the data, so that we have just one way of writing something that's an avenue. Alternatively, what we might actually be interested in doing is encoding the fact that this is an avenue somewhere in the tag set for this particular address. Or, encoding it in some other way. Okay? So, the type of data that we're going to be getting in to our auditing routine is an entire street name. And what we're going to be extracting is just the street type. And, then, taking a look at all of the different street types that are mentioned in the data set that we have. Now the data set that we're going to work with here Is actually only a fraction of the complete Chicago Open street map data set. In this case, we're going to be working with about 20% of the total data set. The entire data set is almost 2 gigabytes. So for text, that's a pretty large data set. Okay, back to the code.



So, as I mentioned, we're going to be getting street names, and what we want to do is recognize types from street names. Now, I'm going to use Python's regular expression module in order to parse out the street types. So, let's take a look at the regular expression that's here. Okay. We're going to match a

sequence of non-white space characters optionally followed by a period. This is to catch abbreviations like ave or st as an abbreviation for street, and this match must occur at the end of the string. And we're going to use this regular expression right here. And each time this function audit street type is going to be parse the street name. We're going to do a match against that string looking for something like ave or avenue or street. Okay. Let's take out more holistic look at this particular piece of code. Now, again, this is just a small portion of the type of auditing that we would want to do, so the way I've set this up is the main routine here is called audit, and what audit is doing is it's looping through this XML file using the Sax parser that's part of the element tree module. From Python. Let's just take a look there. See? And in this case I'm actually using the C implementation of this particular module. So I'd encourage you again to go take a look at the documentation for XML etree, in particular the element tree module. Okay, now what we're going to be doing here is looping through that file. We're going to get one tag at a time here. And, what I'm doing each time is asking the question, is this a street name? Now, if we were doing a full auditing parse, we might actually have multiple things that we're checking here and be generating multiple types of audit data. In our case, what we're going to be doing is essentially creating a record of all street types that we find in this data set. So, let's go ahead and run this.

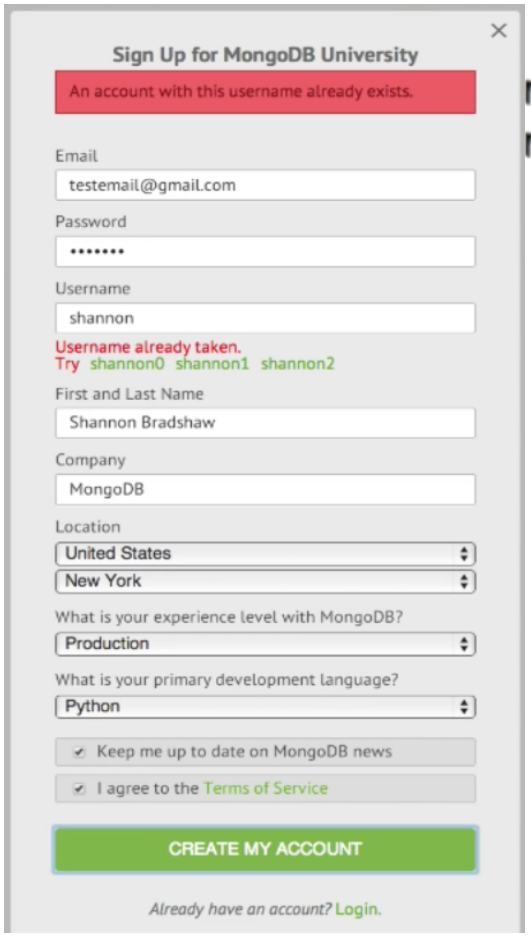
```
street_type_re = re.compile(r'\S+\.?$',
street_types = defaultdict(int)
```

Okay, and here's out output. Now, what I did here, was essentially keep a record of the number of times I saw all the different street types. And my parser recognizes the street type from the street name. We can see that we've got four different forms. Of avenue, the most common one being the full word avenue and then ave as an abbreviation without a period, although there were several that had a period. And again remember this is only 20% of the data type. Okay, the other thing that's nice about auditing the data this way is I would have forgotten about street types like court. For example, and I've actually got three different types of court represented here, and probably also boulevard. And we can see that that also is represented. So what we've done here is essentially audit all of the street types as entered by the human editors over the open street map data. Now what we would do with this particular type of data Is then decide, okay, what type of cleaning is necessary? Do we want to convert all forms of avenue into the full word avenue. And, do something similar with all of the other street types. What type of capitalization do we want? Do we want something that looks like this? Or, do we want just the first letter capitalized? And then finally, we can see that there are lots of things that actually matched our parser, which are not in-fact, street types. So, we need to be careful about that, as we're actually doing the cleaning. So that we don't In fact,

introduce further errors into the process. Okay? So, I'd encourage you to take a look at this code which is shared with you and make sure you understand how it works. The next step then, would be to actually use the audit data. That we have here, in order to figure out some cleaning tasks, for street types. And again, as I mentioned, this is just a small portion of the type of cleaning that we would have to do for this particular data set.

Auditing Validity

Okay let's begin to look at each of our data quality metrics in a little more detail. We'll start by looking at the validity metric. Here we are primarily concerned with individual field values. For example, we might have fields that are mandatory there must be a value there. We might have fields that must have a unique value in each data item. We might have fields with foreign-key constraints. An example might be product records. Each of which must have a reference to a manufacturer. And, that manufacturer must exist in our database. We might have cross-field constraints. What I mean by this is, essentially conditions on multiple fields. So, for example, the start date must come before the end date. Now for most fields, we're also going to be expecting a specific data type. Or, a particular structure of some kind, if we're talking about JSON data, where fields can have values that are themselves dictionaries or arrays. We might also have fields that we expect to have a particular patterns, a pattern we can test using regular expressions. Some fields will have values that we expect to be in a certain range. This might be a numeric range that we can do bounds checking for, or we might be talking about set membership of some kind, so all tee shirts have to have a size field that's small, medium, large, or extra large. Auditing validity is about determining what the constraints are on individual fields and checking to make sure the field values adhere to those constraints.

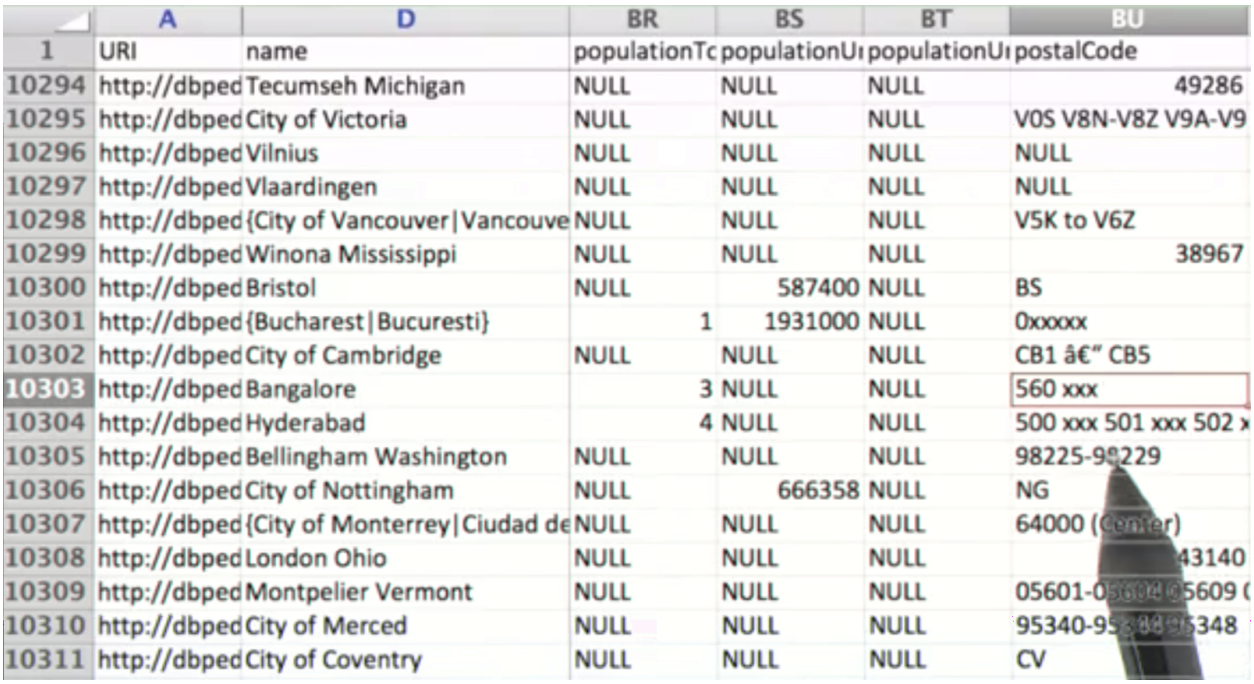


rev.osm
rev.osm

Wikipedia Infobox Dataset

So at this point, I should stop and talk a little bit about the type of data that we'll be working with, in a number of places in the rest of this course. So I'm sure you're aware that Wikipedia,

in addition to the article text for many types of data, also includes info box data. Now, info box data is essentially structure data that complements the subject of the article. In this case, this is an article for the city of Hyderabad and, in addition to a description of the city here and other interesting information about the city, we have data in the form of, population, land area in square kilometers, as well as a number of other useful pieces of data about this city. Now, Wikipedia has info boxes for a lot of different types of data. People, animals, automobiles and so on. An organization called DBpedia has taken the info box data from Wikipedia and produced it as a collection of data sets that we can download. Here's a description of some of the data sets themselves and some of the details about them. Now, in our case, we're going to be working with the city's data set. This data is distributed as a CSV file that we can download. It contains all of the info box data for a particular snapshot in time for all cities described on Wikipedia, for which there is info box data. Now, in order to prepare this data as CSV, the DBPedia folks had to do a little bit of work in encoding it, to get certain types of data values into individual cells. Let's take a look at the city's data set.



	A	D	BR	BS	BT	BU
1	URI	name	populationTc	populationUı	populationUı	postalCode
10294	http://dbped	Tecumseh Michigan	NULL	NULL	NULL	49286
10295	http://dbped	City of Victoria	NULL	NULL	NULL	V0S V8N-V8Z V9A-V9
10296	http://dbped	Vilnius	NULL	NULL	NULL	NULL
10297	http://dbped	Vlaardingen	NULL	NULL	NULL	NULL
10298	http://dbped	{City of Vancouver|Vancouve	NULL	NULL	NULL	V5K to V6Z
10299	http://dbped	Winona Mississippi	NULL	NULL	NULL	38967
10300	http://dbped	Bristol	NULL	587400	NULL	BS
10301	http://dbped	{Bucharest|Bucuresti}	1	1931000	NULL	0xxxxx
10302	http://dbped	City of Cambridge	NULL	NULL	NULL	CB1 â€" CB5
10303	http://dbped	Bangalore	3	NULL	NULL	560 xxx
10304	http://dbped	Hyderabad	4	NULL	NULL	500 xxx 501 xxx 502 x
10305	http://dbped	Bellingham Washington	NULL	NULL	NULL	98225-98229
10306	http://dbped	City of Nottingham	NULL	666358	NULL	NG
10307	http://dbped	{City of Monterrey|Ciudad de	NULL	NULL	NULL	64000 (Center)
10308	http://dbped	London Ohio	NULL	NULL	NULL	43140
10309	http://dbped	Montpelier Vermont	NULL	NULL	NULL	05601-05604 05609 (
10310	http://dbped	City of Merced	NULL	NULL	NULL	95340-95348
10311	http://dbped	City of Coventry	NULL	NULL	NULL	CV

We're going to look here, in Excel, because this data is so dense, we really do need the added structure that the spreadsheet app provides. So here's a row for Hyderabad. And, there are about 20,000 rows in this CSV file; so, lots of cities are represented here. And, we can see that there are lots of null cell values. This is coding in this data set which indicates that there is no value supplied for that particular piece of data. Now, something that's also important to bear in mind is that while this data is actually in pretty good shape, it's entirely human generated. You can edit it by editing the Wikipedia page, as usual. There is some of that data here. So what this means of course, is that there's a lot of dirty data here. Data that's going to need to be cleaned up. So this data set provides a nice example of wrangling data, because we're going to have to decode some of the fields in order to get them into the form that we want. We're going to be looking at transforming these lines of data into JSON documents, and then, eventually, storing them into MongoDB. Let's take a look at a couple of examples of the task that we have ahead

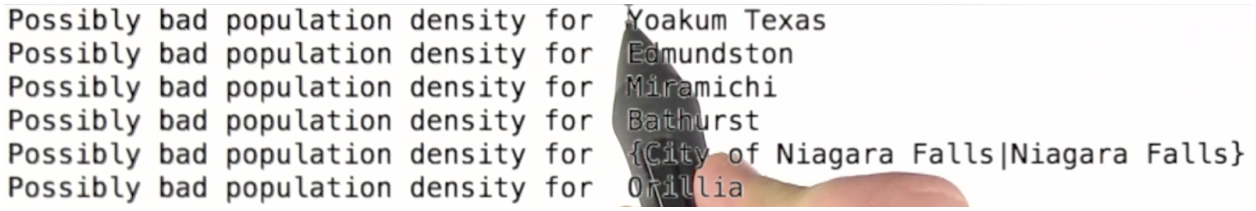
of us. Okay, this is the postal code field, and we can see that there are lots of different formats represented here. Something else that's interesting about this data, is the way they've encoded arrays. Some fields have multiple values stored within them. Here's the utcOffset field. And in this case, we can see that there are two values for the utcOffset for this city and a couple of others here. There's two different values that represent the utcOffset for when the city is on Daylight Savings Time versus when it's not. And you can see that we have some HTML entities represented here, and some Unicode characters, and some other characters that we're going to need to a little work with. So we've got our work cut out for us with this data in terms of pulling it out of the CSV form, parsing it into individuals fields, and cleaning up some of the problematic data that's here. After that, then we'll take a look at getting into MangoDB. So as you can see this data set provides a nice opportunity for us to explore things like auditing a data set, to see where the problems lie. And then thinking about exactly what we should do with fields like this and other structures that we're going to find in this data set. And then finally, we'll take a look at putting this data into MangoDB.

Auditing a Cross-Field Constrain

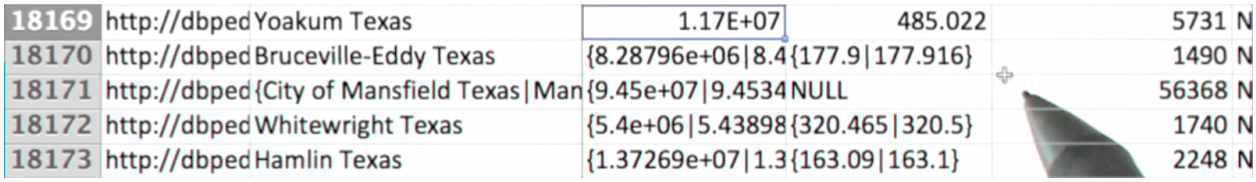
So let's take our conversation about the validity of field values a little further and look at auditing a cross-field constraint. So, remember this is a constraint where we have multiple fields per item, that must be in agreement in some way. Once again, let's look at a Wikipedia page for city. In this case Bern in the country of Switzerland. So here we have a nice example of a city that has values for population, area and one for population density. Any time you have a situation like this, where you can essentially check at least one field against another, you have a nice way of validating field values throughout your entire dataset. So, the cross-field constraint here is that population divided by area gives us the population density. And we'll employ this as one check in auditing our data for the city's collection.

So here I've written a little program to do exactly that. And it does something like calculate the population density based on the individual values for population and area. And compare that to the population density specified within the dataset itself. And we're just going to make sure the difference isn't more than ten to account for rounding errors and that sort of thing. Now, this is just a first pass of this and just an example to show you the type of thing that we might do in

auditing a cross-field constraint. All right, let's run this. We should only see output here. Yep. And we've got a number of bad population densities or possibly bad population densities. So what I usually do in situations like this is, I pick one and I basically just go look at the data and see if I can figure out what's going on. Okay, this one stood out to me, so let's go take a look at that. Now, in the interest of time, I've already pulled this one up in our original dataset. And



I've actually hidden some of the fields here, so it's easier to see the values that we actually care about in this case. So, here's the area field, populationDensity and the populationTotal. Now what we want is for this value divided by this value to be pretty close to this value. And if you look at this, you can see that there's no way that can possibly happen given the size of this. Okay and as I look at the rest of these values here. I also see some values that appear to me to be impossibly large as the area of land for a given city. Especially given that when I look at the Wikipedia page and as I look at Wikipedia pages for other cities, I notice that area is actually measured in square kilometers. So there's no way I have this many square kilometers. However if I look at Yoakum, Texas, I can see that it's value for area is value. If I go back and look at this data and if I interpret this instead of kilometers as not even meters, but millimeters then this



makes sense.
Now from my perspective having a land area specified in millimeters is a little crazy. But I can see reasons why the dbpedia folks might have set things up that way. But if I think about shifting the decimal point for this value appropriately so, that it describes square kilometers, instead of square millimeters. Then I end up with a value that's here. And these values were collected some time ago. So, I'm willing to bet that our problem here, that we're seeing in auditing this cross-field constraint. Is really one of having made a faulty assumption about the land area. And that in fact is what the problem is. And now of course, in order to verify this, we'd want to look through a number of other fields here. And actually maybe do a bit of Googling around to really be sure that is exactly what's going on here. And that it's just an issue of units as opposed to an actual problem with the cross-field constraint, that we know has to exist between these three fields. And that's an example of auditing a cross-field constraint in our Cities dataset.

Auditing Accuracy

Now let's turn our attention to Auditing Accuracy. As I mentioned in the overview, this is something that is difficult to do in a lot of situations because it requires gold standard data. Let's take a look at our city's info box data set for an example. Returning to the Hyderabad wikipedia page. If we scroll down, we can see that country is one of the fields that's included in the info box data for this, and for most countries in this data set. Now it's possible in our data that the countries specified for a city isn't actually a valid country for one reason or another. So in order to audit this country field for accuracy, we're going to need some means of programmatically processing it and comparing and the values that we find to some set of data that we can reliably assume is correct. So, the gold standard data that I'm going to use here is some country code data from the ISO. The International Standards Organization. This is data that pairs a two digit country code with a country name. So in order to verify these country codes, we're essentially going to do a look up of each name we find in this list of country names we can rely on.

Auditing Accuracy 2

Okay, the way I like to go about auditing my data is to write code that performs that kinds of checks that I'm interested in doing. now, you could use something like a piece of ETL software, there are a number. Clover, there are some Python-based ones, PETL is one of those. For me, I find those get in the way a lot and this might just be personal preference. In terms of this class, I think it's better for us to talk about the basics. So what I'd like to do is just give you an idea for the type of process in terms of the codes you'll be writing in order to do auditing and cleaning tasks and we'll just write Python scripts to do this. So in my case, I'm interested in auditing the accuracy of the country field here. So I've written a little script here that is going to follow the process I just described of comparing country field values in our data with this reliable list of country names.

audit_country.py

```
 9 client = MongoClient("mongodb://localhost:27017")
10 db = client.examples
11
12
13 def skip_lines(input_file, skip):
14     for i in range(0, skip):
15         next(input_file)
16
17 def audit_country(input_file):
18     for row in input_file:
19         country = row['country_label']
20         country = country.strip()
21         if (country == "NULL") or (country == ""):
22             continue
23         if db.countries.find({ "name" : country }).count() != 1:
24             print "Not found:", country
25
26 if __name__ == '__main__':
```

Now, this is the type of thing that we would probably want to use a database for. So, we're going to have a little bit of a preview here for the next lesson in that,I'm actually taking advantage of some data stored in mongoDB, in fact, that country codes data, that I've stored in mongoDB. And I'm going to just do a very simple query in this code. That happens right here using the PyMongo module. Which is the Python driver or client library for Mongo DB. Again, what I'm doing is moving through every value in my cities data and executing a query against the database. Asking weather the value I'm finding in the country field is in that list of country names from the ISO. Which I have stored in a collection in mongoDB. Okay so, let's run this and see what we find. Okay, so this just ran through all 20,000 odd cities and here's what we get as country names. That were not found in our list. You can see that in some cases what we've got here is the need to do some further parsing on the country field. Because we've got multiple names listed. And you could imagine why for some cities, there would be two names listed. So, let's just note this. This is just problems with the country field. So, the first thing that we found is that some values are actually arrays. Take a look at this one. Or this one. This is naming a county in Wisconsin, a county in Texas and a county in Florida. So these are examples of the wrong type of value being in the field. A lot of times that is due to something called column shift which is a particular value that got moved one column over. Now, that's probably not in fact what happened here. But the idea is the same. Someone entered a value that would be valid in another field, other than our country field. Then, we have some instances of data that simply needs to be cleaned up a bit.

```
Not found: Waukesha County Wisconsin
Not found: {Lutsk Raion|Ukraine|Volyn Oblast}
Not found: India national cricket team
Not found: Republic of Ireland
Not found: {De facto|De jure}
Not found: {Galmudug|Somalia}
Not found: Georgia (country)
Not found: {Jubaland|Somalia}
Not found: Ancient Egypt
Not found: North Sumatra
Not found: Dnipropetrovsk Oblast
Not found: Republic of Ireland
Not found: Southeast Region Brazil
Not found: North Sumatra
Not found: North Sumatra
Not found: Northeast Region Brazil
Not found: {German Reich|Prussia}
Not found: The_Democratic_Republic_Of_Congo
Not found: East Timor
Not found: {Florida|Miami-Dade County Florida}
Not found: São Tomé and Príncipe
Not found: {Chernivtsi Oblast|Hertsa Raion}
```

So, the parenthesis here is to distinguish the country of Georgia name in the United States. And here you can see we've got an u individual words for this particular country. I would call those exa expressions to come to the rescue. We simply need to account field and clean them up by writing a little regular expression Tha that's sitting there. And then finally, we've got some countries th recognized in our list for one reason or another. There might be language issue. So for these, we need to make a decision, and we want to include these, maybe add them to our gold standard have to make on a situation by situation basis. So, there, we sim countries that we'll need to make a decision about. Okay. So, th example of how we might go about auditing accuracy for a parti

Auditing Completeness

Now completeness can be difficult to assess. As my friend David Silverman is fond of saying, you don't know what you don't know. Now in this discussion we're not talking about individual fields missing from a record. Instead, we're actually talking about missing records. That is to say, trying to figure out when it's an entire record that's gone missing. So the solution here is very similar to the solution for accuracy. Essentially, we need reference data. So, let me give you an example from something I work with on a regular basis. I'm the director of education at MongoDB. And part of my role, I'm responsible for our certification program. Now, we do things a little bit differently than some other technology companies, in that our certification exams are

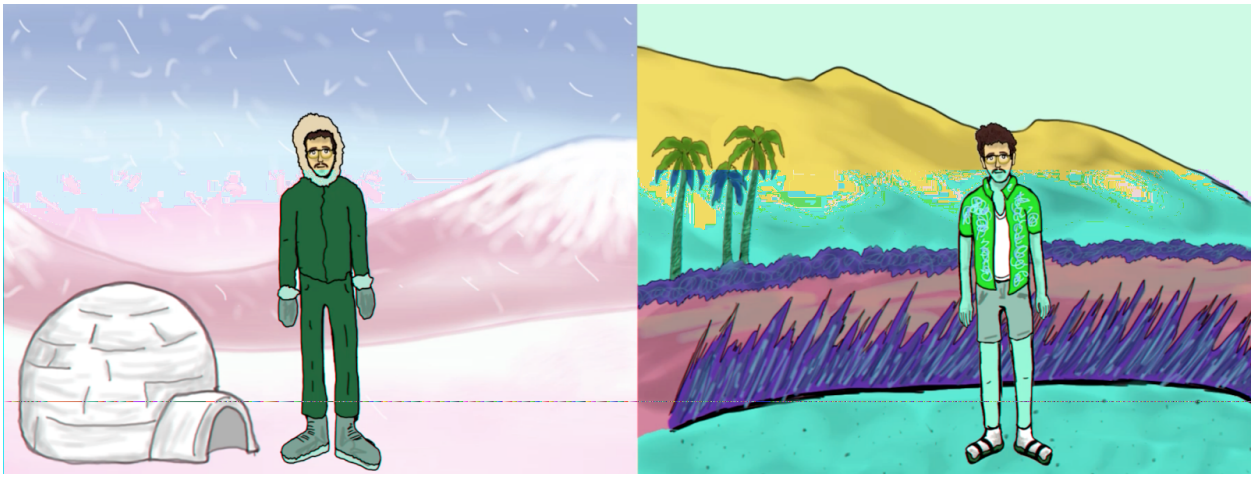
delivered entirely online. So, in addition to a completed exam record for every test taker. In delivering our exams online, we have to have a way of proctoring them to ensure the security of the exam. The way we do that is through a web proctoring solution where capture video of the test taker session. We capture both video of the test taker themselves, using a web cam, as well as screen capture for everything that's going on on the test taker's computer screen as they're taking our exam. What this means, then, is that we have three separate data stores that need to agree with one another on a couple of different things. And here is where we get to the completeness example.

They must agree on the list of test takers, that is to say, if there is a record for a test taker in any one of these databases, there must be a record for that same test taker in the other two. They also need to agree on the duration of the exam session. So the video, here and here, should be approximately the same length and it should agree with the elapsed time that we recorded for a test taker. And, of course, these have to agree approximately. Within sum Epsilon. Now, you're probably thinking what if somebody took an exam and the record doesn't exist in any one of these three databases. Well, that's exactly right. As I mentioned at the beginning of this, this is a difficult problem to address because we don't know what we don't know. In that case using the solution that we just described we would not detect the missing exam record. So, in point of fact, we actually do a couple of things in addition to this to ensure completeness for our exam records. And those are essentially preventative measures to make sure that we can't get into a situation where we haven't captured the appropriate exam data for a given test taker. So, to wrap this up, as with much of data cleaning, the means of auditing completion is situation specific. It really depends on what the data is that you are auditing and what reference sources you have access to.

Auditing Consistency

Inconsistency occurs when two different entries contradict one another. So for example, if a customer is recorded in two different current addresses, we've got a consistency problem. Fixing consistency problems really boils down to the problem of who do I trust the most, or more specifically, which data source do I trust the most? So with our address example, we might ask ourselves which entry was collected most recently. or if our problem is one that's a little more general say just location information. Another type of question we can ask is which collection method is likely to be most reliable? Because one data source might be populated based on

GPS while another is based on user enter data. We might actually have a third that's using IP based location detection and all three of these are contributing location information to our system.



So again, addressing consistency problems is about determining which of our data sources is most likely to be correct. And possibly introducing agreement among our inconsistent records by copying the data from the record we trust the most.

Consistency

Okay, great. Let's do a quiz having to do with data consistency. Now in this case, we're going to look at stock symbols for three different companies over a period of several years. This is an example drawn from the Bad Data Handbook. Spencer Burns, in his chapter, When Data and Reality Don't Match, tells this story having to do with unique identifiers for a few different companies that are publicly traded. So for most of the 20th century, stock symbols for Kmart and Sears were KM and S, respectively. Sprint also, during the time it was a company during the 20th century, had the stock symbol, FON, or FON. So now, let's take a look at what happened with regard to these unique identifiers for companies as we move forward in time. And, what I want you to think about is; what type of challenges might this present to us from a consistency standpoint, and thinking about auditing data and cleaning data.

1	Start	Kmart	Sears	Sprint
2	20th Century	KM	S	FON
3	12/19/2002		S	FON
4	6/10/2003	KMRT	S	FON
5	3/28/2005		SHLD	FON
6	8/15/2005		SHLD	S

So, in 2002, Kmart filed for bankruptcy and was then delisted as a stock. In 2003, they recovered from bankruptcy, were relisted, but this time with the stock symbol KMRT. In 2005, Kmart and Sears merged and so all former shares of Kmart became SHLD, or Sears Holdings, as did Sears stock. And then later that year, Sprint changed its stock symbol from FON to S. So, over a period of several years, three different companies were uniquely identified by five

different stock symbols. And in fact, two companies were, depending on which point in time you're talking about, uniquely identified by the same stock symbol. Now when we talk about these being unique identifiers, we should really be using air quotes. It's this kind of problem that we'll often face, when we are looking at data collected from different points in time. The identifiers for specific entities have a tendency to change. Now this type of data is actually fairly difficult to find. So, in this quiz what we're going to do is look at something similar, having to do with when countries joined the eurozone and what that means for what currency they were using at a given point in time.

Auditing Uniformity

Okay let's talk about our last data quality metric that being uniformity. And we're going to look at auditing in particular field for uniformity. So if you remember, uniformity is about all the values in the field using the same units of measurement. Let's look at an example. So here we're going to work with cities data set again. And what I want to explore here is just one field, that being the latitude field. Now the latitude field is identified in this data set using this particular field name. So let's take a look at some of the auditing tasks we might do here. Now the way that I've organized this code is that we're going to loop through each of the rows in this data file, and again here we're using our tft module in Python. For each row we're going to call this function audit_float_field. So this particular piece of code is something that we can actually use to parse any field that should have a floating point value. In general this is how I like to think about auditing fields in my data sets. I like to think about the things in general that can go wrong with a particular type of data field. And do some auditing for that type and then if I need to, I can write more specific auditing routines to check the values.

```
def audit_float_field(v, counts):
    v = v.strip()
    if v == "NULL":
        counts['nulls'] += 1
    elif v == "":
        counts['empties'] += 1
    elif is_array(v):
        counts['arrays'] += 1
    elif not is_number(v):
        print "Found non number:", v
    else:
        v = float(v)
        if not ((minval < v) and (v < maxval)):
            print "Found out of range value:", v
```

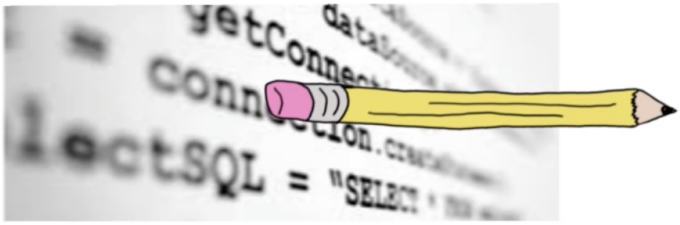
Okay, let's take a look at that audit_float_field function. This is where all of the real work happens here. So, what I'm going to do is I'm going to keep track of the number of nulls that I find, the number of empty fields, if any, and then the number of field values that are actually arrays. And if you remember arrays are encoding using curly braces and vertical bars to separate the individual elements of arrays IN the info box data set. I'm also going to check to make sure that the value is actually a number. And then if it is, I'm going to run a check to make sure that, it falls within the minimum and maximum values, okay? So, this is a way of making sure that it's using the units of measurement that I expect. And if you remember earlier, we saw an example where the area for a city is actually represented using square millimeters as opposed to square kilometers. So, what I'm doing in this particular piece of code, is actually hard coding in some values for this particular field. Now what I would do here, if I wasn't using this as an example for this course, is I would actually treat each of these as command-line parameters that I would input to this script.

Here I'm just going to hard code them in. So, if I wanted to actually use this for a different field what I would do is change the field name and change the min and max values to test for a different float field. Okay. So, going back to our audit_float_field function, again, we're checking for nulls, empties, arrays, any fields that are not in fact a number, once we've made it through all these tests. And then finally, if I get down to here, I've got something that I believe to be a number. What I'm going to do is actually convert it to a floating point value, because of course, all the values coming in are strings, and then I'm going to check its range. Okay, now the range for latitude, the way this data should be encoded is between negative 90 and positive equal to.

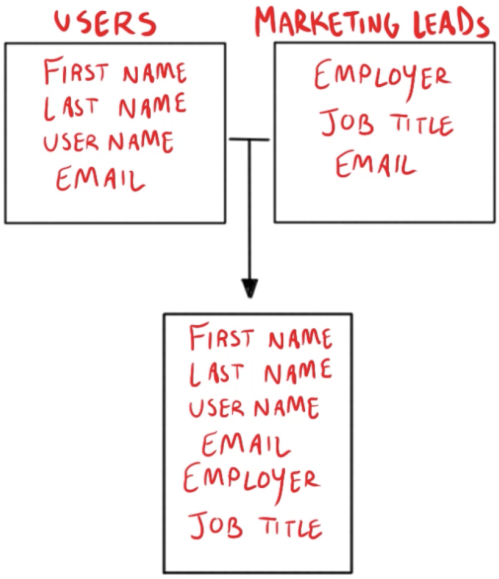
Okay. So let's run this, and see what pops up. Okay? So I found three non numbers. And you can see this looks like an okay latitude value, just expressed in a different type of unit. Total number of cities, that's what I expect. Quite a few nulls, actually. Okay, not much we're going to do about that in this particular example. And quite a few arrays. If I wanted to fully audit this, I would need to take a look at these arrays and see what's going on there. And then I would need to check each of the individual values in those arrays. What I'm more concerned about, in this particular example, are these. Now there are several different ways of representing geographic coordinates. This is three examples where instead of having the raw values for latitude and longitude, we've instead got this type of coordinate, which is actually degrees, minutes and seconds. So a different way of encoding the same information for latitude. If I change this code slightly we'll get a chance to see what the bulk of the values actually look like. Okay, and you can see that they are all values between negative 90 and positive 90 and there we can see a few negative values as well. So, commenting those out, running this again. What's going on with these values? Well, the story could be that these numbers were coded by hand using a different coordinate system, and that's actually why we're seeing this come out rather than this type of number which is what we expect. So this is the type of thing we might see when we're auditing for uniformity. We've got a single field that holds a particular type of data, in this case, latitude values for the location of cities. But there's two different coordinate systems being used here. The decimal degrees latitude and latitude represented in degrees, minutes and seconds. Now, in interest of full disclosure, I actually made the data set dirty by introducing these three values. But this is exactly the type of thing you might expect to see in terms of the same type of value being represented using different units.

```
16.2
26.02
18.82
13.52
23.2
10.1229
12.44
12.66
14.9
num cities: 20229
nulls: 1371
empties: 0
arrays: 2059
education:Examples 10gen$
education:Examples 10gen$ python audit_float_field.py
Found non number: 34 25 00 N
Found non number: 32 22 29 N
Found non number: 53 24 N
num cities: 20229
nulls: 1371
empties: 0
arrays: 2059
education:Examples 10gen$
```

More About Correcting Data



So, in summary, I'd like to say just a little bit more about correcting data. The types of corrections that we typically make to any data set that we're cleaning, usually involve one or more of the following. We're going to be removing typographical errors, or correcting them. There will often be a component in cleaning where we're validating against a known list of entities. We will frequently be cross checking our data with a validated data set of some kind. We might be doing data enhancement, and frequently are, where we're making data a little more complete by adding some related information. So for example, we might have a username, an email address, and first and last name for every entry in a record set. And as part of our cleaning process or part of our process before, we analyze data. We're going to integrate each of those records with marketing data for each person on employer and job title et cetera. Another common thing we do in cleaning data is harmonizing the data. So for example in this case, we would be transforming all abbreviations for street and road to say for example, their full names, or vise versa. Maybe we're going the other direction. Likewise with directions. N and W become north and west, et cetera in addresses. We might also be changing our data set so that it uses a new standard for some set of codes that it contains. So for example, we might be transforming our data set to, instead of two-digit country codes, use three-digit codes by simply mapping each of the two-digit codes onto the appropriate three-digit codes. These are some of the common types of tasks that we do when cleaning data, and again, it's very situation specific. So what we've tried to do throughout this lesson, and what we're going to do in the problem set, is give you examples from a few different situations, to give you some practice with cleaning, so that you have some concrete examples in mind, when you get into your own data wrangling tasks where you need to do some data cleaning.



Conclusion

Okay, we've just wrapped up lesson three. In this lesson, we looked at best practices for auditing and cleaning our data. In the next lesson, we'll introduce you to MongoDB. MongoDB is an open source NoSQL database. It's used around the world for data science problems big and small.