Presentation Talking Points Plan

Should try to have this last a maximum of 40 minutes. Be ruthless about cutting out the non-essential. Highlight all of the stuff that connects and try to get rid of (or minimize) the stuff that doesn't

# Introduction

* Hello everyone! My name is Emily Lynn, I’m a Data and Analytics Consultant for Slalom, specializing in data science. Thank you all for coming tonight, I hope that this talk is helpful for if you have encountered this issue or encounter it in the future, and that you …

# Examples

* So for tonight we are going to put ourselves into the shoes of a character, and Ganesh has graciously volunteered to represent that character (even though he has no idea what he is getting into)
* Ganesh for tonight you are a empire data analyst working for the Sith. You have a fantastic boss
* PUT UP PICTURE OF PALPATINE
* Your boss, the Emperor Palpatine, has tasked you with doing an analysis on rebel heroes so that you can provide recommendations on how to more easily crush any such heroes that will pop up in the future. However, if you do not provide accurate results, you do run the very high risk of being executed with Sith lighning.
* Now you get to work, and you realize that this database has been created by pulling data from several inter-galactic data collection agencies and there seems to be a lot of duplicates in it. You know that with a dataset like that your chances of being struck with lighting are higher, so you start to look at some examples of these duplicates to see if you can devise a plan for how to deal with the problem.
* This is the first example of a duplicate that you find…. Does anyone have any ideas for maybe how we could deal with these kinds of variations?
* **Ganesh how do you feel about your chances of getting struck by lightning?**
* The next duplicate that you come across looks like this…. Anyone have some ideas for dealing with this type of situation?
  + NLP spelling correctors out there
  + Okay so we can deal with this, a little bit more tricky but like not completely impossible
* **Ganesh how do you feel now? (or how does someone else feel about Ganesh’s chances?)**
* The next duplicate that you come across looks like this: crazy
* Now, at this point Ganesh, instead of trying to figure out how to make rules to deal with all of these duplicate patterns is trying his best to actually find a rebel who he can sell empire secrets to in order to join them and get away from his certain demise because his project is doomed.
* I’ve made a list of the issues that we have discussed…These are the kinds of things that our solution is going to need to be able to handle.

# Real World Importance

* This kind of issue does not just occur in the Empire databases but also is also a frequent problem in our lives as well.
* Alright, so maybe your very life is not on the line, but this issue is something that causes legitimate problems for business. Let’s see if this story sounds familiar:
  + You have a question about your car insurance. So you call up customer service to talk about it and they connect you with a representative. They ask you for your name and phone number and policy number and all that so that they can look you up, and then they proceed to answer your question. At the end they say ‘By the way, our company also offers home insurance, is that something that you would be interested in discussing options for. We have these great deals, blah blah blah.’ But you let them know actually you already have home insurance through this company, but by the way that reminds you that you’ve been getting some fees that you don’t think you are supposed to getting from your home insurance and would like to look into that. The rep is like oh sorry, I don’t have that info that is a different department let me connect you with them. So you wait on hold for 10 minutes and then you get on with the home insurance rep. They ask you for you name and phone number and policy number and then let you know that actually your phone number isn’t matching their records, which annoys you because you just had this exact same convo with the previous rep. And it turns out your method of payment had expired, but when they tried to notify by phone of this issue it was the wrong phone number and they didn’t get the message to you before you were charged several fees. Now you are extra frustrated because you did call to change your phone number and payment info, but apparently that only got communicated on the car insurance side and didn’t end up in your home insurance data. There’s nothing the rep can do about it, so you go to spend the next couple of hours working it out with the claims department or something.
  + Show Frustrated Slide
  + This is a great way to lose a customer.
  + What happened in the background is that this company was built from acquiring various other insurance companies and their customer bases, so the customer information is fragmented in separate data silos and because there are variations they can’t piece it together to make a unified 360 view of a customer and they not only annoy customers but also lose a lot of their ability to upsell and crosssell because they don’t really know what coverage their customers have.
* This may seem like a slightly different issue, but is actually the same problem of trying to find records that refer to the same entity but are just slightly different.
* So earlier we learned that this is a really hard problem, but now we realize that this is a problem that is really worth the effort that it takes to solve.
* GO TO GARTNER SLIDE
* I like the way that Gartner put it in an article of their research findings: read quote
  + Whether those ramifications be frustrated customers that leave, or revenue opportunities that are missed, or something else.

# Entity Resolution is the Solution

* So we’ve set up a situation where we have a really hard problem with serious consequences if we don’t solve it. Maybe some business will have some marketing struggles, but the main problem is that Ganesh is going to get hit with Sith lightning if we don’t solve this.
* But it actually turns out that we don’t have to solve by making a comprehensive list of rules. Instead we can take advantage of ML.
* GO TO UNIVERSE OF ML SLIDE
* Universe of ML
* This issue boils down into a classification problem : two records either refer to the same entity (class 1) or they don’t (class 0). So we’re going to walk through how we address this problem from this perspective instead of from a rules-perspective.

# Overall Process Steps

* GO TO PROCESS SLIDE
* This is a high-level view of the different steps that we are going to take a look at. Some of these steps are going to be pretty familiar. The ones that we are going to spend a little bit more time focusing on are:
  + Blocking (which is how we make this problem computationally manageable)
  + similarity metrics (which is how we turn the two textual representations of these records into a single feature vector our model can understand)

# Introducing the Demo

* I’m going to walk through each of these steps, but I also want to do it in the context of and end-to-end process demo with example data.
* OPEN JUPYTER NOTEBOOK
* The package that I’m going to be using to help with this whole process is called Record Linkage. It takes care of a lot of the background data manipulation grunt work so that we can focus on the part that requires our expertise.
* This library actually provides several example datasets for experimenting with ER, so we are going to be working with one of those today. This is a generated dataset with 5000 records of imaginary people, where 4000 are the original records and 1000 are duplicates.
* Here we can see the kind of personal information that we are working with. I will point out that this data was created with Australian addresses, which is why these state abbreviations are not what we are used to seeing 😊

# Data Normalization

* This particular dataset was generated to contain a lot of typos and those kinds of differences, but other than that there is not a lot of extensive data cleaning required. Pretty standard, nothing very specific for ER hear yet.
* But this would be a step where we could address formatting differences. I’ve had to deal with differences in URL formats, phone number formats, email address, addresses, all of that kind of stuff. A lot of time it can just help to take out all whitespaces and punctuation for those kinds of things.
* Ganesh let’s check formatting issues as something data normalization can often handle.

# Prep for Features

* Again the first step of our other data prep is pretty simple, we are just splitting up the date of birth column into separate year, month, and day columns
* But now we get to start bringing in some prep for some more interesting features that we are going to be building later.
* One thing we are going to do is tokenize our address columns. That’s just the fancy way of saying we are going to split them into lists of words. This will let us apply a really cool technique called TF-IDF later on.
* We are also going to do our best to eliminate all of the different ways that names can be spelled.
  + CONNECT – Raise your hand: if you have ever gone to Starbucks and been handed a cup back with your name spelled wrong, you are familiar with all the variations of how your name can be spelled.
  + Similar to our problem where someone can spell Obi-Wan as OB1
* We don’t want those spelling variations to cause problems for us, but we also don’t want to go through and make rules to standardize every name’s spelling. One great way to deal with this issue is to get rid of the names all together and use their phonetic representations instead.
  + There are several algorithms for this such as Soundex, NYSIIS, and match-rating. Here I’m using the metaphone algorithm to do this.
  + Record linkage actually has this algorithm built in that you can take advantage of
* Ganesh you can go ahead and check off the one about different ways of spelling names

# Blocking

* Now if anyone has been thinking about the math of this process, they may have realized that we can pretty quickly run into some hefty wait times with this process. Theoretically, any record could be a duplicate of any other record, so we would need to run this classification on every single possible pair combination. So roughly that is squared the number of records that we have divided by 2.
  + Out dataset of 5000 records : 12.5 million pairs to check
  + For a dataset of 100,000 records : about 5 billion comparisons to do
  + Grows exponentially from there.
* Most of the Empire’s computational resources are going towards building the Death Star, so Ganesh isn’t going to be able to get a hold of any mega compute power to help him out. This is when we want to work smarter not harder. We’re going to try to find a balance between efficiency and missing possible pairs.
* Blocking is our solution for this. We are going to create groups of records based on common attributes and then only compare records within those groups instead of comparing every record to every other record. Let me explain that a little better with some visuals
* Go to PowerPoint Blocking Slide
* Let’s take another look at our rebel hero data that Ganesh has to work with.
* This process is very similar to the group-by clause in a SQL query. You define what you want to group by, and then only compare each record to the other records that are in its group.
* For instance I could say that I’m going to create groups based on race. So only records that share the same race are going to be compared to each other. – for this data that means that only the first two will be compared to each other.
* This grouping really cut down on how many record comparisons we are doing, but for this particular data that’s not super great though because we miss out on all of the duplicates that don’t agree on the race.
* Or I could say that I’m going to create groups based on the area code of the phone number. Only records that agree on the area code of the phone number will be compared to each other. So all of the first 5 records will be compared to each other, but no records are going to be compared with this last one.
* The goal is to find blocking keys that make small enough groups that it significantly reduces the number of record comparisons, but minimize the number of legitimate duplicates that we miss.
* The best way to do this is to combine the results of using multiple blocking keys.
* Go back to the notebook
* For our example dataset we are going to create blocks (or groups) based on the phonetic encodings of the first name. So only records where the first name sounds the same are going to be compared to each other.
* We are going to do the same with the last name, so comparing records where the last name sounds the same.
* And finally we’re going to create blocks based on the suburb. This suburb method is slightly different though, because we are going to use a method called ‘SortedNeighbourhood’. The is like fuzzy blocking 😊 We’re creating groups that have the same suburb and almost the same suburb. This is great for taking typos into account.

# Similarity Metrics

* Can break the glass by warning the audience to hang with me because this is really the core part and it may take a little bit for it to be explained. Set up the tone of ‘I tried to trim this down but there are just so many different options at this point (lowers their expectations to probably longer than it actually is)
  + At important parts where you are breaking the glass you can walk out from behind whatever you are behind to make a connection
* So now we have a reasonable number of record pairs to check to see whether they are duplicates or not.
* Now this next step is probably THE most important step in the whole process. This is when we turn the similarities or differences between two records into a single feature vector of numbers for our model. Because we have to represent this information numerically.
* For every field we need the computer to look at the value in one record and the value in the other record and give us back a number between 0 and 1, where 0 means they are completely different and 1 means they are exactly the same.
* How we calculate that number is going to depend on what makes the most sense for each field. And there are tons of different comparison metrics you can choose from.
* For instance, categorical columns that don’t have typos in them there’s really only two possibilities – they either are the exact same thing or they aren’t. So that is just a binary result – either 0 or 1 but nothing in between. That is how we are going to define similarity when we are comparing state, and the first and last name phonetic encodings between two records.
* However if something can have typos in it, we’re instead going to look at how close the two values are. There are tons of metrics for this: Levenshtein distance, Jaro distance, Jaro-Winkler distance, Hamming distance, gap distances, Lee distance, there are just tons of them. But the key is to pick one that emphasizes the important information for that field.
* For instance, the Jaro distance is going to count how many characters match between two strings, and characters are considered matching if there are roughly in the same place. This is great for minor typos so we are going to use this one when comparing suburbs.
* But because of the way Jaro only loosely cares if the character is roughly in the right location it means it is not a good metric to use if order is really important. If you move a number 3 spots over in a postcode that is a significant difference in postcode. So instead for fields where order is important we’re going to use the levenshtein distance which counts how many insertions, deletions, or replacements it would take to turn one string into the other.
* So this fuzzy logic is what is going to handle our typo problem. So Ganesh let’s check typos off of our list. What do we have left?
* Both of these, however, are going to be bad at handling abbreviations (like St. instead of Street in addresses) or our optional words problem (like Mr. in names) because those kinds of changes are pretty significant to the length of the string.
* But these are problems Ganesh needs to solve in his data!
* To do this we are going to use a super useful method called Soft TF-IDF. Basically what TF-IDF does is it takes in all of the words that are present for a field across all of the records. So we are using this method with our address fields, so it is going to take in all of those individual words that we tokenized earlier. Its then going to give each word a weight based on how unique it is in the dataset. So for address, Street and its abbreviation St. are both probably going to show up a lot. Because of that, TF-IDF is going to give them almost no significance when comparing fields, and is instead going to give all of the significance to words that don’t appear as often. So this will fix issues like Salutations (like Mr. and Mrs.) in names, LLC or Inc. in company names, common words and their abbreviations in addresses, all of that kind of stuff.
* So Ganesh let’s go ahead and check off optional words and abbreviations from our list.
* Alright, so once we apply a similarity metric to each attribute, for every pair we are going to end up with a feature vector of what those calculated similarity metrics are. One of the last issues we have to handle is missing data. If data is missing from one record and not the other, we obviously can’t calculate how similar the values are. But we don’t want to just give that feature a 0 because that looks exactly the same as if they were completely different values. So the last thing we are going to do is create columns that indicate whether or not values were missing from the records to provide the model with that information.
* So Ganesh let’s go ahead and check off missing data from our list. Do we have anything left on our list?

# Basic Score

* We finished the list! We’ve fixed a lot of our problems already actually. Technically you could just average all of these similarity values and come up with like an overall similarity value for the pair and just define a cutoff value for what counts as a duplicate. I mean this has taken a long time already right we might just want to stop hear right?
* What do you think?
* Run cell with gif of emperor shooting lightning
* The Emperor is not pleased, and I’ll show you why.
* This is what is called calculating a ‘naïve’ score. Or we can call it a ‘basic’ score if we don’t want to insult the empire analyst 😊 Because up until this point we actually haven’t done any machine learning we’ve just gotten really smart with what records we compare and we’ve chosen smart ways of comparing records.
* Let’s take a look and see what it would look like if we tapped out at this point.
* Fortunately this package that provided us this dataset also provided us the answers. So we can use the answers to test out this theory.
* Defining our cutoff would already require a lot of trial and error. But no matter where we do it we are going to miss a lot of matches or could a lot of distinct records as matches.
* The technical term for this area is the ‘mushy middle’. All of these matches in the mushy middle are the ones that we would miss/get wrong if we were to just choose a similarity score cutoff. And this dataset is actually less mushy that probably most real-life datasets.
* I’ve seen tutorials that do this.
* So we are going to go the extra mile.

# Labeling

* If we are going to do ML we will need ground truth. A dataset of records and a label of whether they are duplicates or not is not going to just be sitting around at a company, so 99% of the time you are going to have to do this labeling work.
* For the sake of time I’m not going to talk about this part. There are some special considerations and ways that you can significantly speed this up, so if anyone is interested we can take a look at the end or something like that.
* But I do want to show one of my favorite things that recordlinkage provides that makes it so much easier to do this labeling. They’ve create a Github pages hosted labeling tool that let’s you upload a file of data, click some buttons to indicate whether records match or not, and then download the results in a format that works seamlessly with the rest of its functions. In the past I’ve made files of 50 pairs, handed out five files each to 5 different people gave them a couple of days because it doesn’t take super long, and then BAM I have 1,250 ground truth examples. So even though this is not the type of labeled data that is typically readily available in a company, its not too bad to make manually.
* If they are 100% similar, we can automatically just define them as being duplicates.
* If they are really dissimilar, we can also automatically define them as being non-duplicates.
* BAM we already have 700 something ground truth examples.
* For the rest of them we’re going to use our basic score to slice the list of pairs into sections and pull equally from these sections.
* Alright so I’m going to spare us the tedious task of manually creating ground truth. Our empire analyst can just get some droids to do it.

# Train the Model

* Alright we’ve finally gotten to the easy part where we actually train this model!
* So this is me just loading some ground truth for us to use. We have X ground truth examples. But a lot of those we didn’t actually have to manually label because of being smart about using out naïve score, but we can go back and look at those specifics at the end if that is interesting to anybody.
* We’re just going to go basic and classic with a logistic regression model, and these are the results that we actually end up with.
* Now I wan to make this clear – These results are a little bit ridiculous. You could cut yourself on this PR curve. We’ve used a dataset that is more straightforward for the sake of demo brevity, but a lot of real-world datasets are going to be more messy. So I’m not promising X results every time you do this process.
* Now I do want to point out that I print the Accuracy here, but accuracy is actually a terrible metric to use for this situation,
* Does anyone know why?
* because the classes are so skewed. This is the same reason why instead of a typical ROC curve we are going to want to look at a Precision Recall curve when evaluating the model instead.
* And of course depending on the business use case you may want to adjust the confidence threshold that the model is using to emphasize the precision or recall based upon the particular situation.
* This is the point where depending on your results you will likely have to go tweak one or multiple of the previous steps right in order to improve them. Put in general, for our particular slightly unrealistic situation we seem like we are pretty good at this point.
* But Ganesh how are you feeling now about your chances of being able to handle the Emperor’s request?
* I count that as success for the evening 😊

# Clustering

* So we’ve determine what records are pairs with our ML model, our very final icing on the cake step is to pull all of those duplicate records together so that the business can see all of the data for this customer in one place and hopefully do some work to consolidate that information into a single complete golden record.
* We've taken the problem and made it manageable. This is the point when companies will probably want to have some hands on to do the canonicalization because this is important information about customers that they may want to have eyes on. This could be an optional ending and may not be a good idea to introduce new info at this point unless you set this up at the beginning as the end goal. The business case is not just finding the dupes but actually doing something with them - the canonicalization. The business doesn't just care about the dedupes. This is the resolution of the story - DS is the hero! This is a definite place of interaction between the DS and the domain expert where you can sit down and talk about these interesting situations and what can be learned from them. Learn from each other and then that is something that feeds into the iterative process.