this is the fourth video in a larger series on full stack data science in the previous video of the series I discussed how we can make data pipelines for machine learning projects here I'll discuss the next stage in the ml pipeline which is how we can use data to build AI Solutions I'll start with a highlevel overview and then dive into a Hands-On example with python code and if you're new here welcome I'm Shaw I make videos about data science and Entrepreneurship and if you enjoyed this content please consider subscribing that's a great no cost way you can support me in all the videos that I make although we can draw parallels between traditional software development and machine learning development there are several key differences that are important to keep in mind the first and most fundamental is that in traditional software development the rules and the logic that make up the program are explicitly written into the computer by the programmer however when it comes to machine learning computers aren't told what to do explicitly but rather the rules or the instructions of the program are learned from data directly while this allows us to build ml solutions for things we could never write traditional software for such as text generation or autonomous driving this indirect way of programming computers gives rise to a few other key differences for one the behavior of traditional software systems are typically predictable in other words given any input for a traditional Software System you can typically know what the output is going to be on the other hand the behavior of machine Learning Systems is a bit more unpredictable you don't always know how the system will react to particular edge cases no matter how many tests you come up with to evaluate your system there will always be examples that you can't take into consideration because there are an infinite number of them another key difference is that traditional software systems are usually interpretable meaning you can usually have an intuitive understanding of how a software system took any given input and generated a specific output on the other hand machine learning systems are often uninterpretable or at least they're not interpretable in the same way that traditional software systems are so even though a machine Learning System can often generate better performance than a traditional Software System that often comes at the cost of interpretability and then finally traditional software development typically has a linear development cycle or at least a clear development cycle in other words projects can progress in a predictable manner on the other hand developing machine Learning Systems is often iterative and progress might be made in a nonlinear type of way while these differences create several Downstream consequences and how we should think about machine learning development as opposed to traditional software development the main thing I want to focus on here is the role of experimentation the way I see it this is what makes data science closer to something a scientist might do rather than an engineer more specifically scientists typically have hypotheses that they'll test against experiments while Engineers are typically implementing a given design of course it's not always this black and white in practice but experimenting with multiple potential Solutions is a key role of a data scientist so what this typically looks like is represented by this flow chart here so what this is representing is that we have the real world which is full of things that are happening some things we care about some things we don't care about what we do when we want to build ml Solutions is we collect data about some of the things that we care about in the real world and then we make that data available so that we can develop a machine learning solution with it once we have a candidate solution we can evaluate the efficacy or the value of that solution typically this results in a set of feedback loops so you might evaluate a solution see that the performance isn't so great so you go back and you tweak some parameters and you evaluate it again and then you tweak some more parameters and you keep going in this feedback loop and of course this Loop might even be automated you may exhaustively search a bunch of different parameters and still not get the results that you want so you decide to go back and change the data set that you're using for your solution tion development and perhaps repeat this whole process finally you might realize that the data that you have available isn't sufficient to develop Your solution so you go back to the real world and you re-evaluate the data that you need this is why developing ml Solutions is iterative and often nonlinear because you might go through hundreds of iterations of your solution before finally realizing that you weren't collecting sufficient data and then once you grab one key variable for example and pass it into your model you find that you finally get the performance that you need and the value is generated so to make this a bit more concrete let's look at a specific example let's say we wanted to develop a semantic search system this is something I've talked about in a couple previous videos of the series including the one on Rag and the one on text embeddings but if you're not familiar with semantic search the basic idea is that we start with a set of documents and then we take these documents and we generate numerical representations of them which we call text embeddings then what we can do is develop a Search tool where a user can type in a query we can generate a numerical representation of this query and then we can evaluate which documents are closest to the user's query and return them as search results and it's called semantic search because rather than using specific keywords in the user's query the meaning of the query and the meaning of the documents are captured by these numerical representations since I have a video All About text edings I won't go into the details here but I'll link that video in case you want to learn more while this might seem like a pretty straightforward idea take documents generate text embeddings and then do some kind of similarity score between the query and all the different documents there are several design choices that come up When developing this system them for example given documents documents have a lot of text in them so what text do we want to use for example if these are blog articles do we just want to use the title do we just want to use the first paragraph of the blog do we want to use the entire blog another one is should we chunk the text if we're talking about a Blog it could include a lot of different information where one paragraph is relevant to a potential users's query but the rest of the document is irrelevant this may result in our semantic surge to be a crude approximation of the underlying information in the documents another question is should we summarize the text you have a long document maybe you want to summarize it just so you capture the key information before passing it into an embedding model but of course there's more what embedding model do you want to choose there are several readily available models both open source models and closed Source models also should we embed multiple parts of a document so if you have an article again do you want to embed the title and the B body of the document separately and then maybe combine them in some way and then talking about the Search tool like how do you want to measure the distance between a query and all the different documents how should we filter results you have millions of documents it might be a good idea to narrow down the candidates before applying the semantic search because it's a bit more computationally expensive and then should we use meta tags you want to add tags to documents to help with this filtering process so all that to say there are countless design choices that come up When developing any machine learning solution and even everything I discussed here is far from an exhaustive list so to make this even more concrete let's look at a real world example of building a semantic search system here I'm going to walk through a project that I'm currently building to perform semantic search over all of my YouTube videos and this project has been the focus of this larger Series where in the previous video we built the data pipeline for this project we started with the data source which was the YouTube API we saw how we can build a data pipeline for this project I extracted information about all my YouTube videos from the YouTube API I did some light Transformations and then I loaded them into a data store specifically a paret file in this video I'm going to walk through the experimentation piece of building this semantic Search tool so we're going to take that paret file file which includes things like the video's ID its title and transcript we're going to generate text embeddings and then we'll build a Search tool with a user interface and here there are a few design choices that I will experiment with specifically whether we should base the search on the video's title its transcripts or both picking an embedding model from three op- Source options and then finally defining the metric or how we're going to define the similarity between the query and all the different videos and there will be five options of that looking through this if we have three options time three options Time 5 options these are 45 different options for this semantic search system and of course these aren't things that we're going to hardcode one by one I'll show how we can automatically generate all of these Solutions and objectively compare them to one another using an evaluation metric with that highle overview of what we're going to do I'm going to jump into the code which is available on the GitHub linked here and I'll also put it in the description and comment section below so before jumping into the code let's just see what the final product looks like by the end of this we'll have a user interface like this where we can type in a query and then it'll spit out responses the formatting doesn't look great cuz it's just a PC but we can see if I type in something like llm it'll return a bunch of videos from my channel as well as links to them so that's pretty cool and then we can search something else what are fat tails and then we go we get all my videos on fat tailedness let's see how can I build a semantic search system all right so this is the perfect video to return cuz I literally walk through it in this video we'll come back to this and play around with it a bit more but anyway I'm going to walk through three different notebooks all available on the GitHub repository the first one is going to be the experimentation piece where we're going to Loop through all all 45 different options and compare them all to each other using an evaluation metric once we figured out which of the 45 options is best we'll create a video index based on that configuration and then finally we'll write the search function and create the user interface starting from the top first I import polers which helps us handle the data structures and polers if you're unfamiliar is basically like pandas but it's much faster and is gaining popularity rapidly the project was a good excuse for me to try out polers and so far I've enjoyed the experience then we import sentence Transformers which has a handful of open-source text embedding models we can use and then we import some distance metrics from sklearn the distance metrics will allow us to evaluate how similar a user's query is to each video in the data set we'll import numpy to work with the matrices that we get from the search function and then I import M plot lib which I may or may not use but this is a great thing to have whenever you're doing any sort of experimentation of machine learning models so you can plot things like histograms and Scatter Plots to compare the performance of different solutions first we load the data like any other machine learning project the way I do it here is I have two data sets one is a data set of the transcripts saved in video- transcripts. par it's a data set containing all of my YouTube videos and YouTube shorts so has all my video IDs the dates they were posted the title of the content and the transcript this is just the head we can also look at the shape so I have 83 videos very small data set by ml standards but it took a long time to make those 83 videos next we have this evaluation data set which consists of two columns one is example query and the other is the ground truth video associated with that query the point of this evaluation data set is to give us a way to objectively compare multiple potential solutions to one another so whether you're training a model from scratch or you're using a model off the shelf like we're doing in this example you need to have an evaluation data set so you can effectively compare multiple candidate Solutions together we can also look at the shape of this data set and so we see we have 64 examples next I'm doing some data preparation what I'm doing here is I'm going to l Loop through each title and transcript in the original data frame so each of these titles and each of these transcripts and I'm going to Loop through three different embedding models available in the sentence Transformers Library so two different columns with three different models gives us six possible configurations in this chunk of code I Loop through every possible combination so you'll have the title with these three models and then you'll have the transcript with these three models so six possible combinations I'll Loop through each one and generate the embedding so what that looks like is a nested for Loop so I have a for Loop for the model name and I have a for Loop for the column names I'm going to store everything in a dictionary so I initialize that here and now just walking through this code first we Define the embedding model that we want to use we set model equal to sentence Transformers model name and then once we have the model we can generate an embedding for a particular column here I Define a key so we have a unique identifier for each element in the dictionary and then in this line of code I'll use the model to generate the text embeddings for every piece of text in that column for example if we're encoding the title this will take the title column of the data frame convert it to a list and then pass it into this encode function and spit out a array of all the embeddings finally we'll store the key name and embedding array in in the dictionary so the key name is just going to be a unique ID it'll be the model name with the column name and then we'll have the embedding array for that combination if we look at the embedding array that's going to be 83 by 768 so we have 83 videos and then the text embedding has 768 Dimensions so that's where this number comes from and of course each embedding model will be different another thing we can look at is this text embedding dictionary view of that we'll see that we have the model Name appended by the column that we're embedding and then we'll have a numpy array with all the numbers associated with each text embedding so if we look at this one specifically we see it's a numpy array and then we can look at its shape and then we see this one is 83x 384 notice that different embedding models will have different embedding Dimensions so this one is actually smaller than the other one which would have been this small model yeah so this model has 768 while the other one has 364 or whatever it was I already forgot going back to this time function this is really handy when it comes to doing these experiments because it'll automatically spit out the time it took to run this line of code here this is helpful because it can allow us to get a rough idea of the computational cost of each of these configurations so we can see that generating embeddings for the transcripts tends to take longer than for just the titles with this case being an exception maybe there's some kind of startup cost with running the first one and then these models tend to have different costs associated with them and the reason is that they actually get bigger and bigger another thing I'll share is that if we go to the sentence Transformers documentation they have a handful of pre-trained models here let's see all mini LM six yeah okay so this is one that we're using it's actually the smallest one and we can see that it's 80 megab while the largest one that we're using multi QA mpet the largest one we're using is more than five times as large at 420 megabytes so these are all important things to take into consideration not just the performance of the solution but the computational cost associated with it because that plays a role as well and then another thing going back this code might be difficult to read or seem a little complicated because we have these nested for loops and we don't really know the model names and column names they're stored in this list here some may have the inclination to want to hardcode all of these things for example just taking this line of code of defining the model name and then this line of code of generating the embedding array and then copy pasting something like this we'll take the model name embedding array doing this and then tweaking it and then repeating that for this and then so on and so forth while in a sense this might be simpler when it comes to doing experimentation across multiple potential Solutions this is an absolute nightmare because say you take this to your team or you read an article talking about how great this other model is if you wanted to go back and change your code it's a lot to keep track of cuz now you got to change it here and then maybe two cells down you use the model name again and then you got to think about keeping track of this and then if you're copy pasting inputs like this you're bound to make a typo and then it's going to cause issues down the line that is the number one reason why I could not recommend enough to write your code something like this where you have somewhere where you basically Define all the different options that you're trying to play with and then just let the code run its magic below and print out all the results that you need to see manually going in and tweaking code blocks here is going to inevitably lead to errors and this is just something I learned the hard way in grad school where I would train a model present it to the research group and they're like oh that's amazing but what if you tweaked this and what if you tried this and then I'm like oh okay so I'd go back but then my code wasn't written like this a lot of manual tweaking and then I would mess things up and things would stop running then I would finally get it working and take it back to the group and then they would come up with some other suggestions and so writing it this way allows you to iterate much faster and helps you avoid a lot of headaches that was a bit of a lecture there but it's super important next block of code basically doing the same thing but instead of embedding the titles and the transcripts for each YouTube video doing it for each of the queries in the evaluation data set this code is a bit simpler since we don't have to iterate through the colum names but it's exactly the same then we move on to evaluating the different search methods here I Define a handful of functions which we can just skip for now and I'll return back to them as we come across them in the code but here I'm doing a similar thing as before I'm listing all the different ways we can evaluate the similarity between the query and a particular video here I list three different distance metrics from pyit learn then two different similarity metrics from the sentence Transformers Library we're going to evaluate all possible combinations of model columns to embed and distance metrics or similarity scores so again this is 45 different combinations even if you could have hardcoded the last six combinations do not hardc code 45 different configurations just write the four Loops similar situation here we're going to Loop through the models here I'm grabbing the text embeddings for all 64 queries in the evaluation data set so I stored them all in this query embedding dick if we look at this thing we see it's a numpy array and then we'll have a row for each query and then we'll have a column for each embedding Dimension then we're going to Loop through all the text columns and we're going to pull the text embeddings for that particular column first we'll start with the title this is going to pull the text embeddings of the titles for every one of the videos looking at that this will also a numpy array but we see that the number of rows is 83 because I have 83 videos and then finally we have a third for Loop because we're going to Loop through each of the distance metrics this will get us this disc object which we can use to compute pairwise distances for all the videos and all the queries so this final thing will be an array of distances we can look at the shape notice that there 83 rows for corresponding to 83 videos and 64 columns corresponding to 64 queries in the evaluation data set each element of this array will be the distance between the E video and the J query for example if we looked at the very first element this would be the distance between the first query in our evaluation data set and the first video in our video index we're going to use this ARG sort function from numpy to sort each of the columns and so if we go back to the disc array we have 83 rows and 64 columns so if we sort each column we're going to rank the videos from smallest distance to largest distance for each of the 64 queries since it's ARG sword instead of returning the ordered values themselves it's going to return the index of the values in ascending order next I Define a method name and this is essentially like we did before where we had a unique name for each combination of model and column but here we're going to combine the model name the column name and the distance name so each of the 45 configurations for this Search tool has a unique name so here I use a function that I defined called evaluate true rankings which evaluates the ranking of the ground truth in other words for a given query we have 83 possible videos to return but only one ground truth in the evaluation data set what this function does is that it returns earns the ranking of the ground Truth for each of the 64 queries that function is defined here and I won't walk through this because I feel like that might get too far into the weeds but if you're curious the code is available on GitHub but we can look at the shape of this thing we can see that it's essentially a one-dimensional array with a ranking value for each of the 64 queries we can see for the first query the ground truth was in the third position for the second query the ground truth was in the zeroth position so it was the number one ranking and so on and so forth and so what I do here is I convert this whole thing to a list and then I append it to the method name so basically a Val list is just going to be one giant list of all the rankings with the first element being the method name and then I store that in another list called eval results so this eval results will be a list of lists where each element is a list corresponding to a particular configuration so we can't use shape cuz it's a list but we'll look at the length and we see yes there are 45 elements 45 elements for the 45 possible combinations this is where a little hard coding comes in because the distance metrics are from pyit learn while the similarity scores I'm importing from the sentence Transformers Library so so since the syntax is a bit different I have to write a different script for that of course this part is copy pasted essentially so I could have been a bit more clever in how I wrote this code but in this specific case I thought it was easier to just leave it how it is like this the one thing I did here which people might come after me for is I dynamically defined this line of code using this syntax and then I executed that command using the exact function command is just a string which looks like a piece of python code so we're defining this distance array as the minus of the similarity score between the embedding array and query embedding and the reason I put minus is that since this is a similarity score it's the inverse of a distance score so in other words if two things are close together a distance score will be small but a similarity score will be large so instead of changing this ARG sort to go the other direction I just add this minus sign so it reverses the order and then the code will be exactly the same we'll sort the indexes like before we'll Define a method name we'll extract the ranking of the ground Truth for each of the queries and then we'll store it in the eval list and then store that list in the eval results list and then here I basically do the same exact thing but it's a little different because I'm embedding the titles and the transcripts while before I was embedding either the tit or the transcripts here I embed both and then it's a lot of the same stuff but here's the key difference when I do the pairwise distance I compute the distance between the title embedding and the query embedding as well as the transcript embedding and the query embedding and then I add those two distance arrays to each other and then repeat the same process so we've seen this chunk of code for a third time now so that's a good indication I should have wrote a function to do this but here we are then I do a similar thing for the similarity scores and so this is the downside of automatically generating code and running it is it's kind of hard to read this line here so we can just run it and take a clear look at it here we Define the distance array as the minus of the similarity score between the title embedding and the query embedding minus the similarity score between the transcript embedding and the query embedding again we have to do that because this similarity score will either be the cosine similarity or the DOT score and we have to add the minus sign to turn the similarity metric into a distance metric and again magnitudes don't matter it's just the ranking that matters which we get from this block of code here which we've now seen a fourth time so this definitely should have been a function and then just some fanciness happening here so maybe this is why I didn't do it as a function because title transcript changes as well and then we have to adjust the similarity name to make the method name come out good I changed the underscore and doore and cosign similarity to a hyphen to make this a little easier to read after that arduous process we've generated 45 different configurations of this Search tool and so everything is stored in this list called eval results which should have 45 Elements which it does but all this information in a list is kind of hard to access so let's store it in a data frame to make it easier to make sense of so to do that I Define a schema for the data frame I do this programmatically where our data frame is going to end up having six 5 columns where the First Column will correspond to the method name which we generated programmatically and the rest of the columns will correspond to the rank of that particular query for that particular method so now you can imagine we're going to have 45 rows in this data frame for each configuration and then we'll have a column corresponding to each query and then the element of the data frame will be the ranking of the ground truth search result for that query using that method so to make that a bit more concrete it looks something like this we have the method names in this column here we have the ranking of the ground Truth for every single query in the evaluation data set as columns so with this first method here let's just print the name so we can see what it is for this first method it's using the model all mini LM L6 B2 it's embedding the title and it uses the ukian distance between the query and the title embedding to rank the search result and then using that method the ground truth was the zero search result so that indicates perfect performance using this metric then we repeat that for every single query and every single search method next I'm going to create two summary statistics so specifically the mean rank of the ground Truth for a particular method and then if the ground truth result appears in the top three results or is the number one result so so this gives us three summary statistics which I add to the results data frame with these two lines of code and then I'll create a new data frame called DF summary that just includes the summary statistics and doesn't have the more granular performance metrics shown here we can look at this summary data frame from three different perspectives first we can rank it by the mean ranking of the ground truth and so this first method had the best performance along this evaluation strategy where the ground truth was usually either the zeroth or the first search result so this method was using all mini LM L6 V2 which was our smallest model it was using both the titles text embeddings and the transcripts text embeddings and it used the Manhattan distance metric and so a Manhattan distance instead of ukian distance which is like the direct path between two points on a graph the Manhattan distance travels along a particular particular axis so distances are computed along grids so the shortest path along a particular grid one thing that is kind of expected is that title and transcript combined together has the best performance and we can actually see that a lot of these results have both the title in the transcript as text embeddings but what's somewhat surprising is that the smallest model had the best performance as opposed to a bigger embedding model two other views is instead of ranking by the mean ground truth ranking we can rank it by the number of top one search results we can see that actually four methods had the ground truth in the number one result and so this was again using this smallest model but these didn't include the transcript they just included the title which is interesting and then they all Ed different distance measures so this one used ninian distance this one was the cosine similarity this one was the doore essentially all three of these methods are equivalent which is very interesting finally we can look at this summary table according to the number of times the ground truth appeared in the top three so again we get three methods that had similar performance but these were all different than what we saw before so now what seems to perform best is our second largest model which is multi QA distill bird poost V1 where they all embedded both the title and the transcript but then used different similarity scores so notice that there was no one method that dominated all others for instance this first method outperformed this fifth method in terms of the number in the top three but this method did better than this method in terms of number in top one similarly even though this method outperformed this method down here in terms of number of top one this method down here outperformed the method up top in terms of the average ranking of the ground truth so this is kind of where the art comes in and you often synthesize this information in your own head to pick out the best strategy and of course you can make this more objective where you give particular weights to each of these evaluation scores so maybe the average ranking of the ground truth is the most important evaluation metric you want to use you'll give this ranking more weight as opposed to this ranking another thing you might do is give more weight to a smaller model as opposed to a bigger model like this one multi QA m pet base. V1 due to the computational cost and the storage cost of a larger model so in this specific situation I went with this model here for two main reasons one I feel the average ranking of the ground truth is a good evaluation metric to base things on and it did pretty well in terms of this number in top three evaluation metric where it was basically in second place and a lot of times you don't need the number one search result