* Let me just pull up a Jupyter notebook and show you how you could do this with a library called langchain.
* I realize this is probably not the IDE most developers are used to working in, and I’m going to be working in Python which may or may not be your language of choice, but just know that the library we’re working with, langchain, is natively supported in JavaScript as well. There are some packages that let you access via C# or Java, but those sometimes are not the most up-to-date. But definitely something you can use in JavaScript.
* Create instance of model
  + So in langchain, models like the gpt-4o behind ChatGPT are just objects, and interacting with the model is just a method.
  + So let me create an instance of that GPT-4o model. And then this temperature parameter here is just telling the model not to get too creative. I like to have this set to 0 to start out so that results are a bit more consistent.
  + This is going to be using the ChatOpenAI class since we are using an OpenAI model. But lanchain has support for pretty much any llm worth using.
  + Once we have this model instance we can just use the .invoke() method to send the model a message/prompt and get back the response
  + Let’s just test it out in the most basic way and ask it for a joke…
  + That sounds a lot like a Dad joke you would make 😉
* Use the image in the prompt
  + We actually want to do something a bit more fancy, which is send the model an image to analyze. To do that, we need to first encode the image in base64.
  + Here’s a nifty function to do that
  + [copy and paste encoding function]
  + Let’s also get a function to display the image to remind us what we’re working with
  + [copy and paste display image function]
  + [copy and paste the path and display the image]
  + Alright so let’s go ahead and encode this. This is what it looks like encoded, just for reference. All ready for us to send to the model.
  + Now multi-modal messages do require a bit more structure in the prompt. So we need to create a message object. This is going to be a HumanMessage object since it’s a message from us to the model.
  + The content of this message/prompt is going to be a list of dictionaries, each of which is a part of the message. So the first one is going to be our actual text explaining to the model what we want it to do. And the next one is going to be the actual encoded image.
  + …Let’s see what we get back.
  + And there ya go, there’s you tags pulled out from the image
* Alright I can see that, let’s make it structured then
  + Is there are particular structure that you would like?
  + Okay we can do JSON, and I’ll actually take it a step further and create an object from the response as well.
  + So let’s define that object that we want. As long as we’re getting information from the model let’s ask it for a description as well as that list of tags.
  + Now we could just aks the model nicely to format its response as a json. But to make sure it does it correctly you really have to be precise and thorough in your formatting instructions. Luckily langchain makes this easy and does it for us with what are called ‘parsers’
  + So let’s use our class definition to create a JSON parser, and a pydantic object parser.
  + Then we can just use the ‘get\_format\_instructions()’ method to generate instructions that we add to our prompt. You can see it explains the structure based on our class definition, and actually gives the model examples of good and bad formatting.
  + Now we pass these instructions in as a new part of our message
  + [Copy and paste the new HumanMessage code]
  + Output looks close to what we want, but as a final step to get it from a string to the right type we pass the response into the parser invoke function and we get back the right structure that we were looking for.
  + Here’s the json
  + And then here’s the data as an object, and we can look at the tags attribute to get those tags we wanted.
  + And let me just put that image back up here as a reminder of what we were doing
* Let’s test how these tags actually would perform
  + I have a whole folder of bird photos tagged with this model just lying around on my laptop (for some reason 😊). Let’s use that to simulate and image search based on tags.
  + Let’s load that file
  + And here’s a function to search those images based on the tags.
  + Let’s start with the obvious tag, just to see what images we’re working with…
  + [search with ‘bird’]
  + Let’s try a bit more specific one to see if we can find our image we were working with before
  + [search with ‘parrot’]
  + Awesome. Now, we know users are rarely satisfied with basic functionality like this, let’s instead use another tag another tag we know we had on our photo
  + [search with ‘funny’]
  + Cool, looks like this was the only one tagged as funny. But what if our user uses a synonym instead of the actual tag?
  + [search with ‘comical’]
  + Or how about ‘silly’ or ‘goofy’?
  + So doesn’t look like we come up with any photos for those.
  + This really shows the limitations of this process.
  + We also can’t really extend this process. What is we wanted to actually use an image search this database for images that are similar? Can’t do that either.
  + But….I can let you in on another piece of data science power that will make this much better. But you have to promise not to let anyone know I told you, because then I’d have to kill you 😊
* Vector Explanation
  + This other piece of data science power is something that we use as the backbone several branches of data science like natural language process and computer vision, and that is vectors. Vectors are a tool that, when used alongside LLMs, really take everything to a new level.
  + So to get this new power you need to bare with me for just a minute as I take us back to high school geometry class. After this you can actually tell your geometry teacher you used it in real life!
  + I know you didn’t wake up this morning expecting to talk about the cartesian coordinate system, but this is for those of us who haven’t been in high school for awhile (and those of us who REALLY haven’t been in high school for awhile).
  + So if you remember, when we define a point in 2-dimensional coordinate space, we define it with 2 numbers – which are how far you have to travel along the X and Y axis to get there. So this purple spot is 3 steps along the x axis, and 4 steps along the y axis. If these coordinates represent actual space, this might be like going 3 steps East and 4 steps North.
  + Same goes for 3D space, we just add another axis and locations are now represented by 3 numbers, one for each axis.
  + Alright, but why are we talking about this? Well these coordinates are actually vectors – a list of numbers that represents a location in this coordinate space.
  + Well what is instead of X and Y we said that the X axis represents the spectrum of ‘good story’ and the Y axis represents the spectrum of ‘good acting’? Now this is a whole new ‘meaning space’. Now I can put different movies at different locations in this space based upon where they fall on those two spectrums. In 3D space we could say another axis is ‘good graphics’ and add that number to the locations of the data points as well.
  + Suddenly these axis are representing features of these data points. The more features we have, the more ‘dimensions’ our space has, and the more numbers we use to represent the location of different things in the space.
  + We can’t really visualize beyond 3 dimensions but this is kind of what that would look like. Once you do this, things that are similar are going to end up closer to each other in the meaning space. So you can see kitten is closer to cat than it is to dog, and on a completely different part of the space than apple. And we can actually measure how close one point is to another using a quick calculation called the cosine distance or cosine similarity.
  + This is really powerful we have the ability to turn a lot of unstructured data into vectors – images, text, videos, etc. That process of converting something into a vector is called ‘embedding’. And once all of these are represented as vectors in the same meaning space, we are suddenly working with a structured representation of their meaning instead of the previous representation where the structure was a lot more hard to work with.
* So let me show you want that means for our image search.
  + We can use a model to turn all of our images into vectors. I’m going to be using an open source version of OpenAI’s CLIP model, which is able to convert both text and images into vectors in the same meaning space.
  + If you look at the model, its super cool, its got a vision transformer, with a convolutional neural network…
  + Let’s just use this model like we use the LLM model before – as an object with methods that we can use.
  + I’ve gone ahead and already vectored our whole set of images. Now if we want to search those images based upon a user’s text search all we have to do is turn that search text into a vector in the meaning space, and see which images vectors are closest to it.
  + Let’s go ahead and load in those vectors…
  + Just to give you an idea of what these vectors look like [print one] they actually have 1024 numbers, which are 1024 features about the image, which technically means we’re working with a 1024-dimensional space!
  + Now we embed the text that says what we are looking for…
  + [Show vector] so this is what the vector looks like for ‘funny bird’
  + Let’s create that function to find which image vectors are closest to this search text…
  + And then here’s just a function to display those images, with their cosine similarity score based on the query text.
  + Let’s see what we get for ‘funny bird’
  + Looks like our image does come up, and we actually find some other images that match that never showed up when we used the ‘funny’ tag before! For each image you can see the similarity score, and the lower down the list we see the lower the score and the less the photo matches our search text.
  + Let’s see what happens if we use a different wording
  + [use ‘silly bird’]
  + We still get those same images, so now instead of being constrained by a finite set of tags, we’re actually working with a representation of the data that takes into account the meaning instead of just specific words.
  + This can even handle ideas that would never even end up in tags.
  + [use ‘bad hair day’]
* Reverse image search
  + And its flexible, because as long as we have that query vector it doesn’t matter where it came from. So I can actually do the reverse image search without changing anything about this setup. All I have to do is instead of the search vector being made from a text string, now its just made from an image.
  + So say we wanted to find photos of this specific type of tucan. Not just any tucan – this specific one. This is an important distinction if we were working with thousands of images.
  + I embed this image, and use it as the search vector…
  + Now you can see that we have a lot of pictures of tucans up at the top, but the ones that are the most like the original are first because they are the most similar to the original query vector.