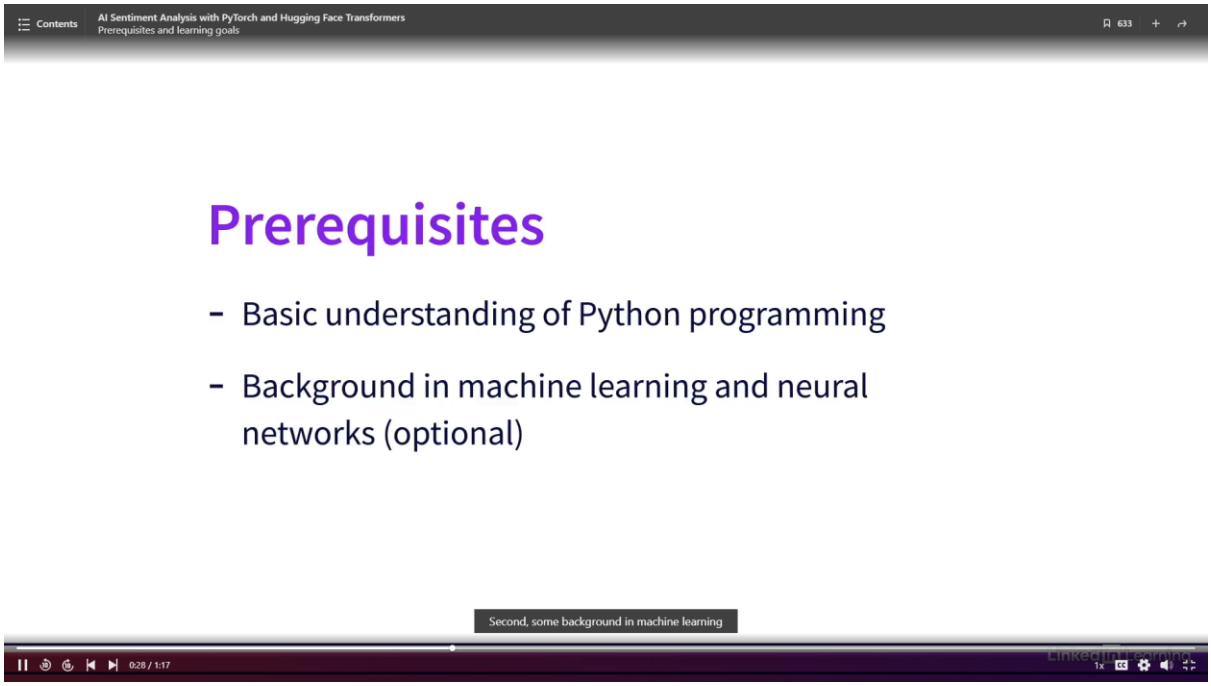
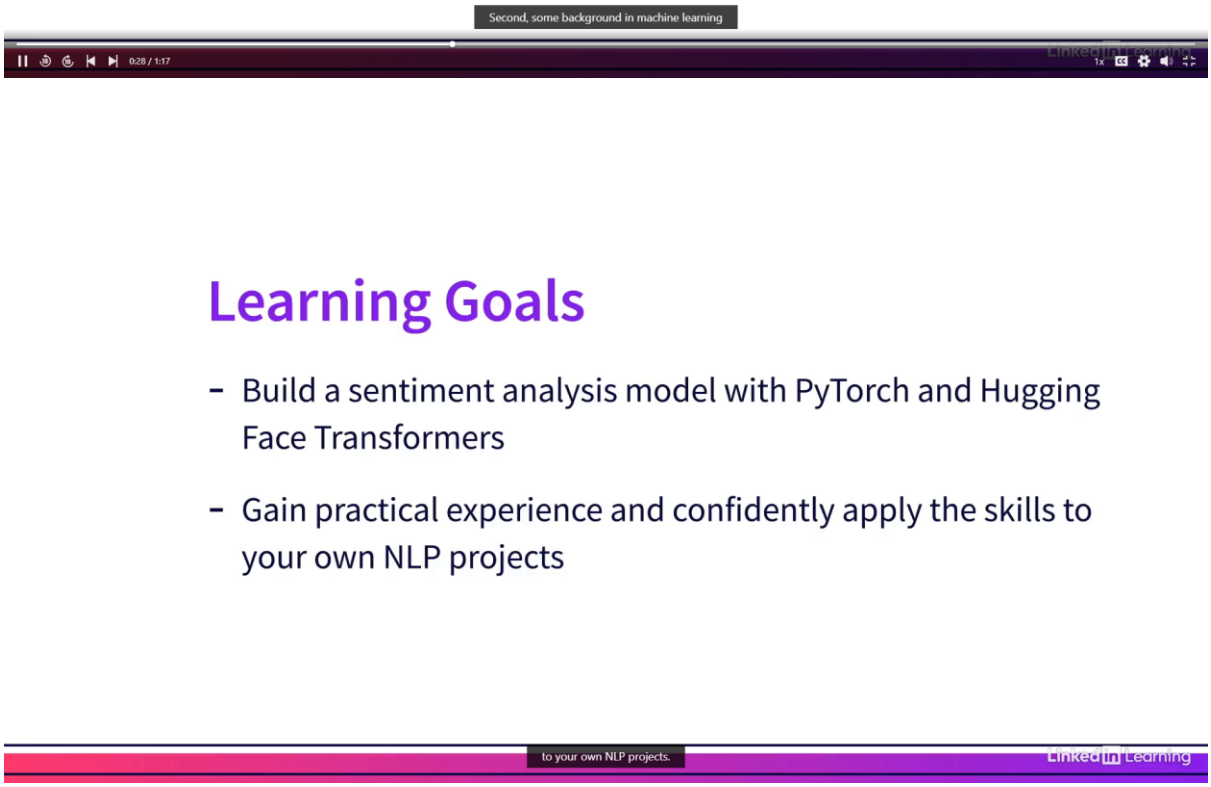
AI Sentiment Analysis with PyTorch and Hugging Face Transformers

Prerequisites and learning goals

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Before we get into the details of sentiment analysis, let's go over the prerequisites and learning goals for this course. First, a basic understanding of Python programming will be helpful. We'll explain the code line by line in later chapters, so being familiar with Python syntax will make it easier to follow along. Second, some background in machine learning and neural networks is preferred, though not required. We'll introduce key concepts throughout the course. The goal of this course is to build a sentiment analysis model with PyTorch and Hugging Face Transformers to identify sentiment in text. By the end, you'll gain practical experience and confidently apply the skills you've learned to your own NLP projects. In the overview section of the course, you'll find an exercise files folder. Be sure to download it so you can follow along with the coding sessions. We'll guide you on how to use them later. Get ready to learn, code, and create.



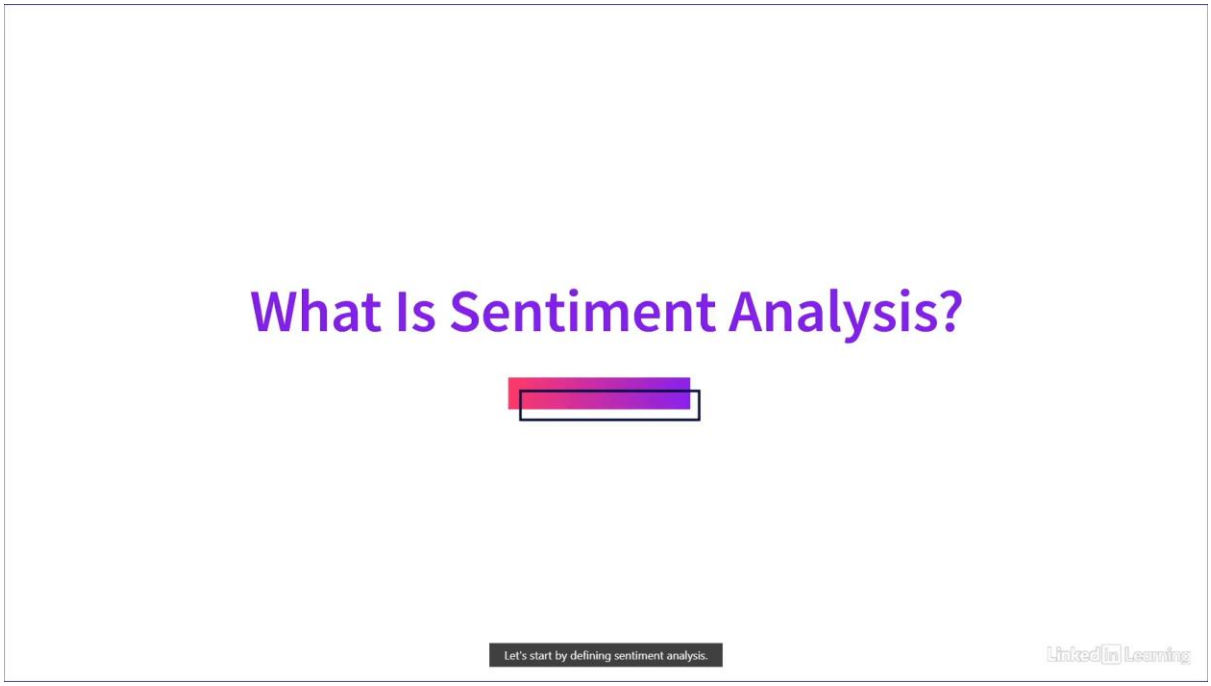


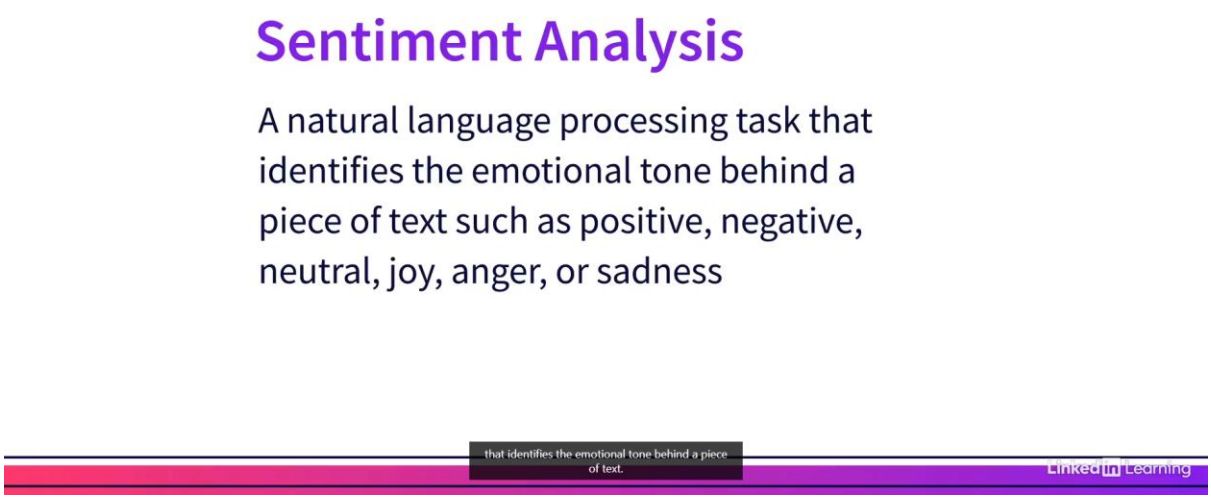
What is sentiment analysis?

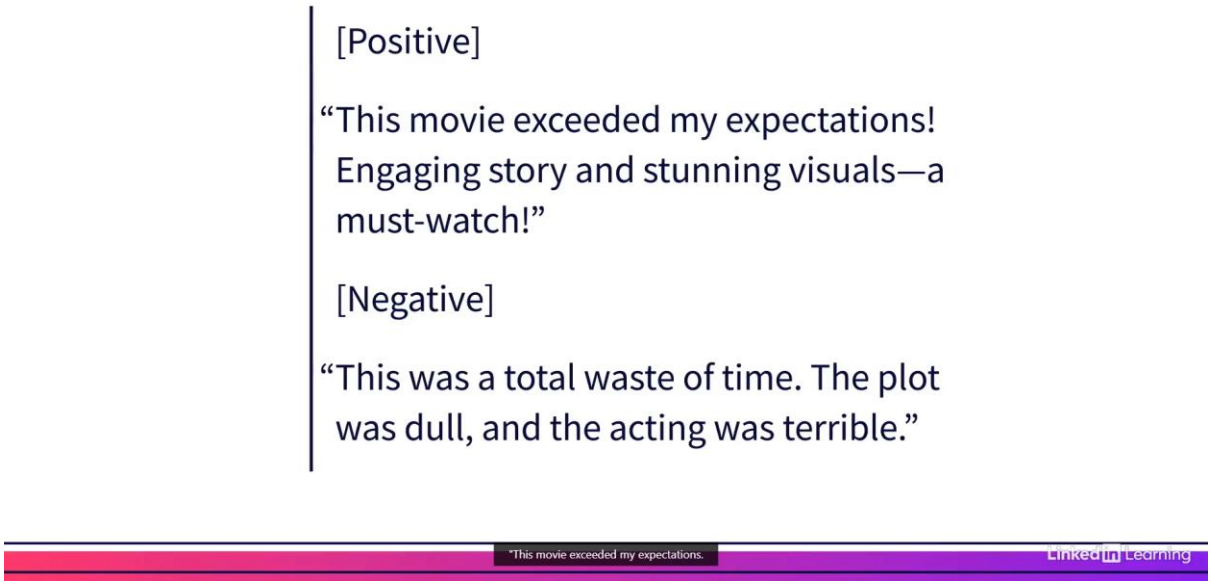
Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Imagine scrolling through social media or reading Amazon reviews. People are constantly sharing their thoughts and sentiments. Let's start by defining sentiment analysis. Sentiment analysis is a natural language processing task that identifies the emotional tone behind a piece

of text. It determines whether a sentiment is positive, negative, or neutral. More advanced models can even recognize complex emotions like joy, anger, or sadness. Imagine reading a movie review that says, "This movie exceeded my expectations. Engaging story and stunning visuals. A must watch." Sentiment analysis would classify this as positive. On the other hand, a review like, "This was a total waste of time. The plot was dull and the acting was terrible," would be categorized as negative. This ability to interpret text-based emotions is incredibly valuable. For example, a movie studio could analyze thousands of reviews in seconds to gauge audience reactions. Up next, we'll take a look at some real world applications of sentiment analysis and see how it brings data to life.





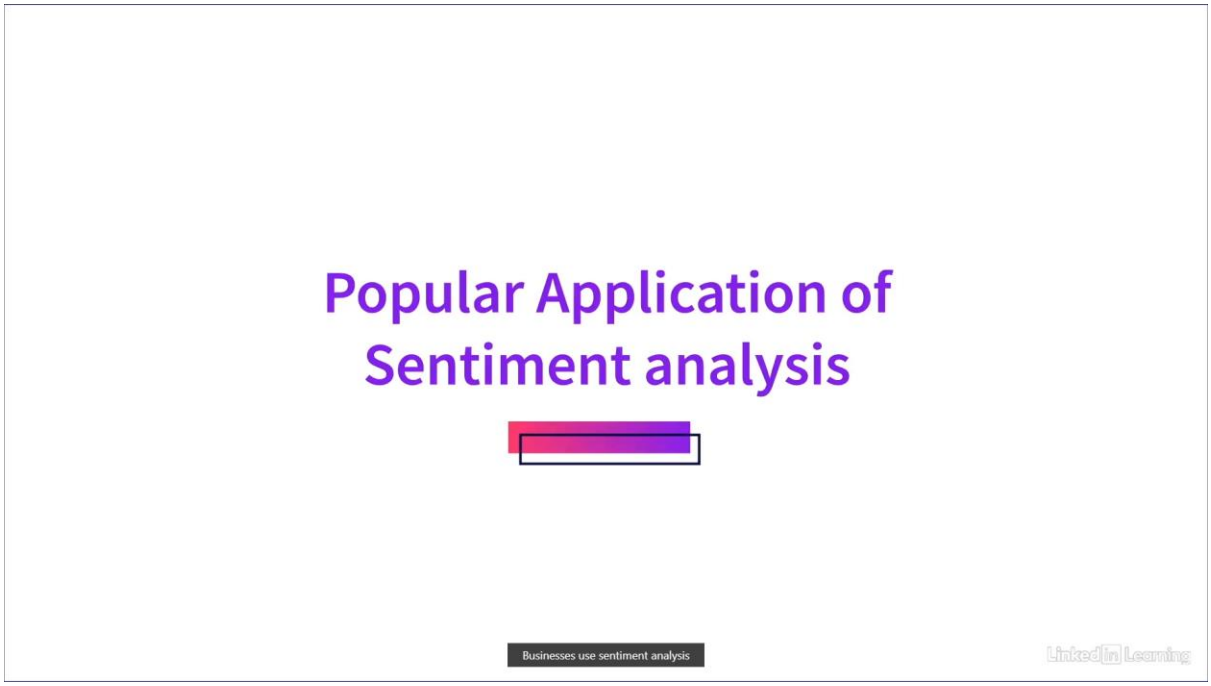


Popular applications of sentiment analysis

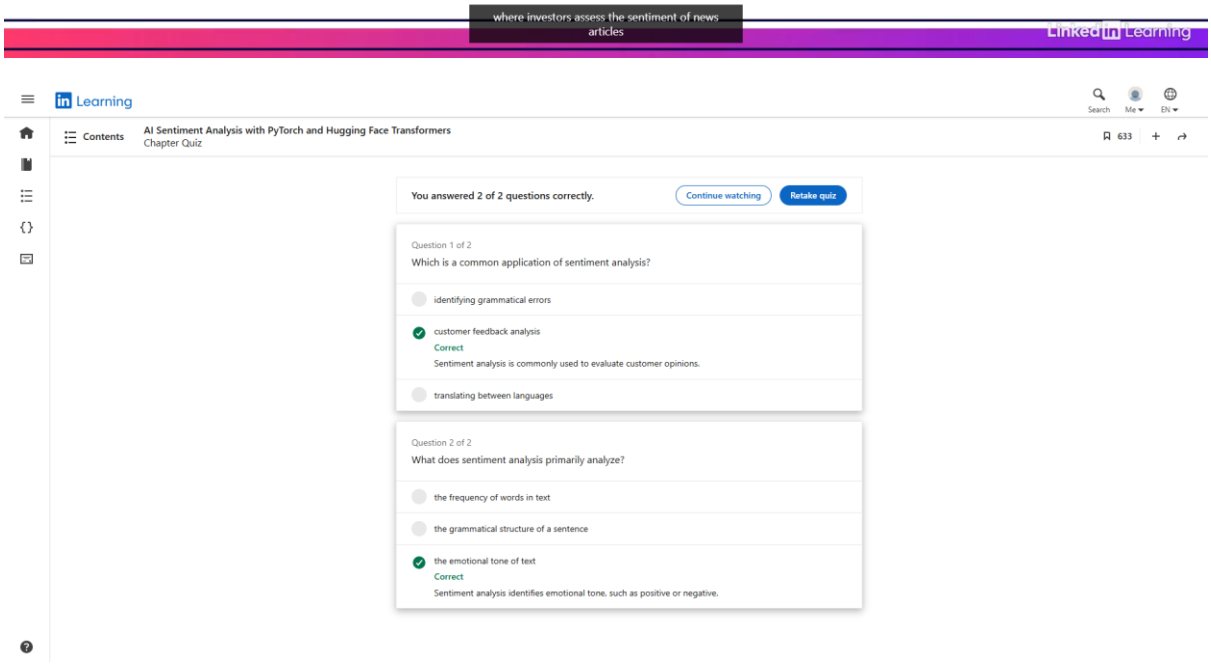
Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Have you ever left a review for a product or service? Businesses use sentiment analysis to process countless reviews like yours, gaining insights into customer satisfaction. Here are some real world applications in

sentiment analysis. One major application is customer feedback analysis. Companies apply sentiment analysis to reviews, surveys, and feedback to better understand customer opinions and make adjustments accordingly. This helps improve customer loyalty. Sentiment analysis is also widely used in social media marketing. Brands monitor public sentiment about their campaigns and products by analyzing tweets, posts, and comments. For example, they can quickly determine whether a movie is receiving more compliments or criticism after its release. In politics, it is used for public opinion analysis to understand how people perceive certain policies, debates, or candidates. Another growing area is the financial market where investors assess the sentiment of news articles or social media posts to predict financial market trends. As you can see, sentiment analysis is a powerful tool with diverse applications. In this course, we'll focus on analyzing the sentiment of movie reviews, but the skills you learn can be applied to many other fields.



- Customer feedback analysis
- Social media marketing
- Public opinion analysis
- Financial market analysis

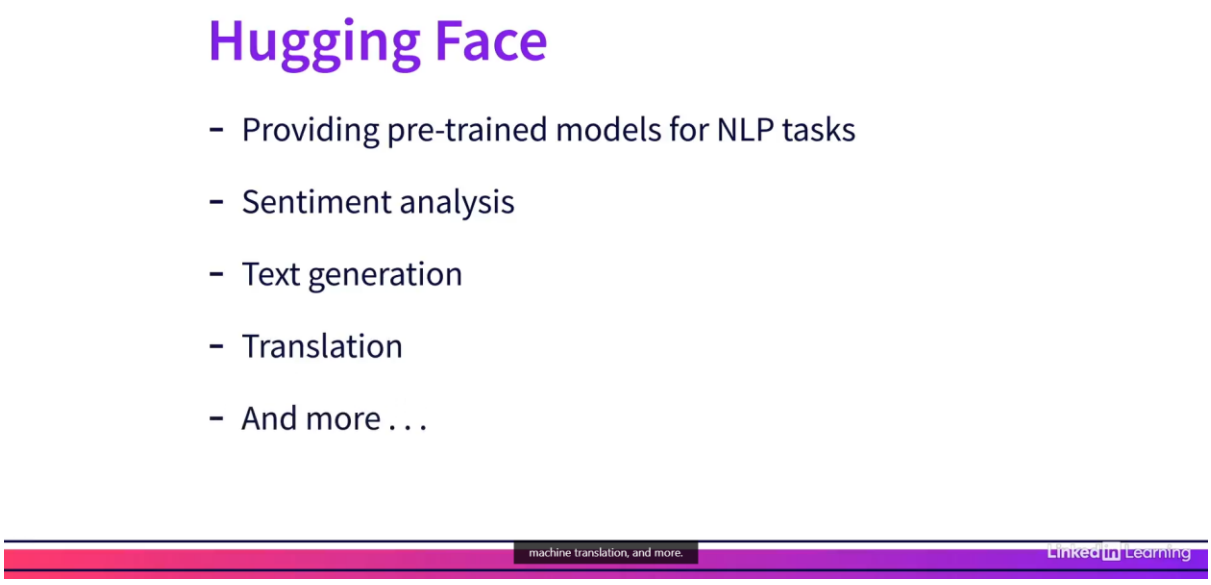


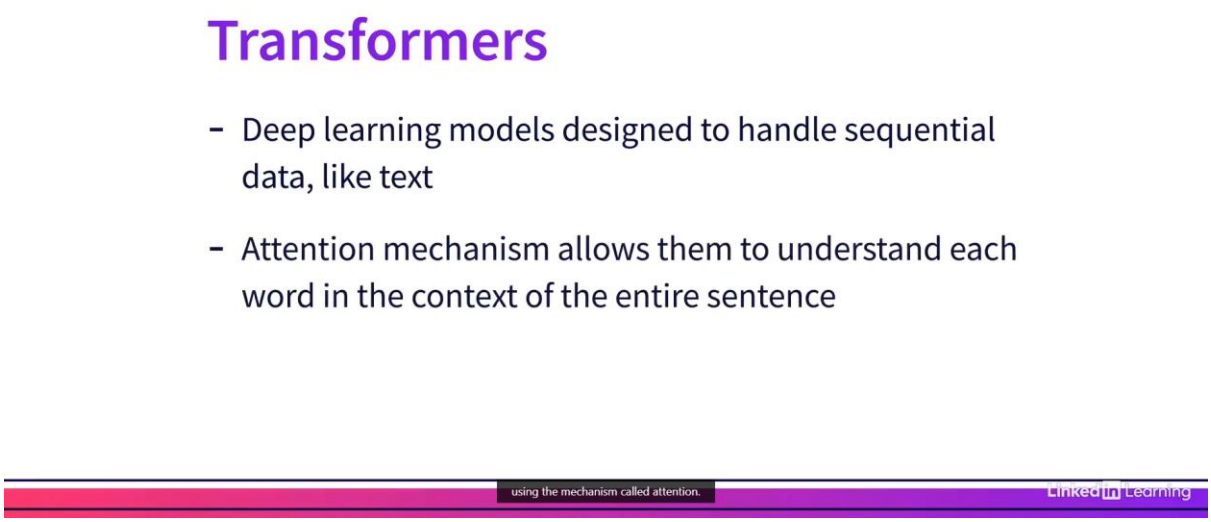
Overview of Hugging Face Transformers

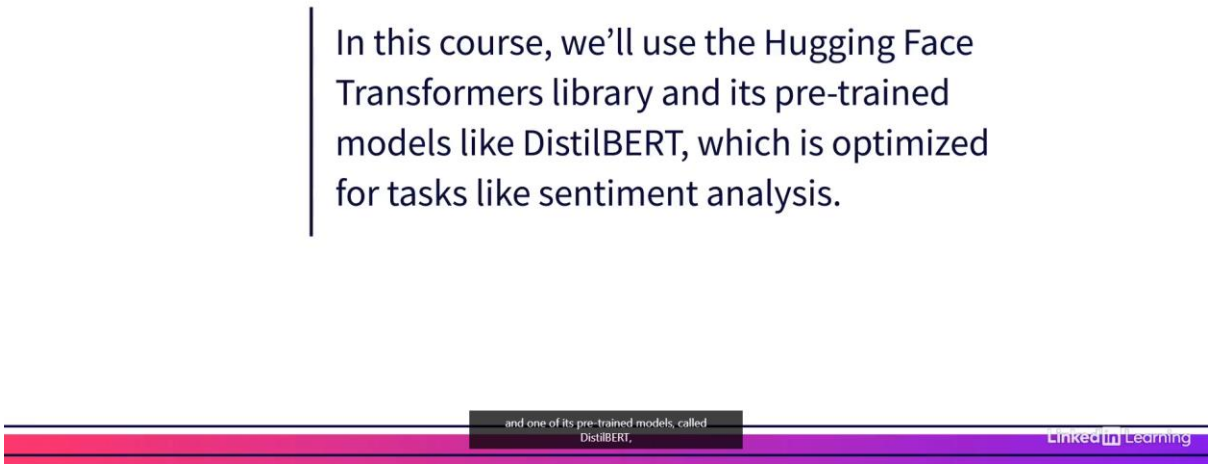
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll introduce Hugging Face Transformers. Hugging Face is an AI company that provides an open source platform for natural language processing. It has revolutionized NLP by offering pre-trained models that can be easily fine tuned for tasks like sentiment analysis, text generation, machine translation, and more. What are

transformers? Transformers are deep learning models designed to handle sequential data, like text, using the mechanism called attention. Unlike traditional NLP models that focus on limited context, transformers understand each word in the context of the entire sentence. Hugging Face has made these powerful models accessible to everyone with user-friendly libraries that save you from the complexities of building models. In this course, we'll use Hugging Face Transformers library and one of its pre-trained models, called DistilBERT, for our sentiment analysis project.





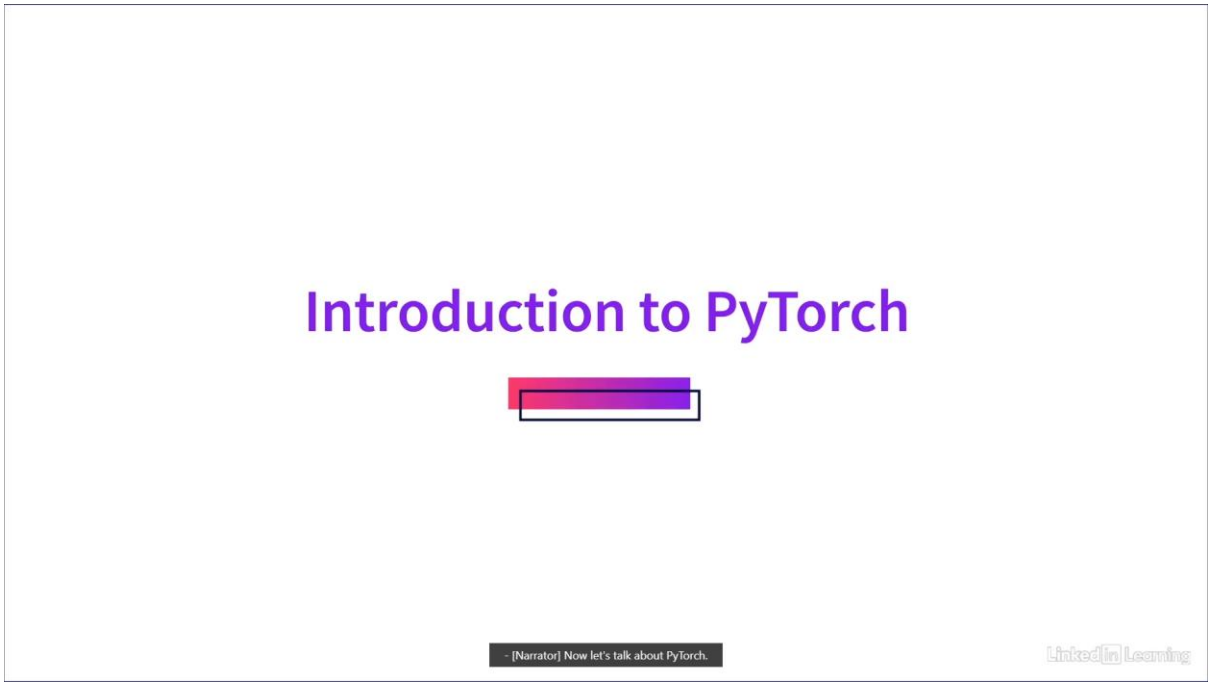


Introduction to PyTorch

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Now let's talk about PyTorch. PyTorch is a powerful, open source, deep learning framework used in research and production environments. It delivers great performance while being easy to use. In this course, we use PyTorch to build and train our neural network. It handles all the underlying

computations for us. Here's the exciting part. PyTorch integrates seamlessly with Hugging Face Transformers Library. Hugging Face provides pre-trained models like DistilBERT for sentiment analysis. PyTorch serves as the backend, managing data flow and optimization. Hugging Face Transformers Library uses PyTorch for model training, inference, and fine tuning. In short, PyTorch is the engine and Hugging Face is the toolkit that provides pre-trained models. In the next video, we'll be setting up Hugging Face Transformers, PyTorch, and other libraries for the project. Let's dive into the code.



What Is PyTorch?

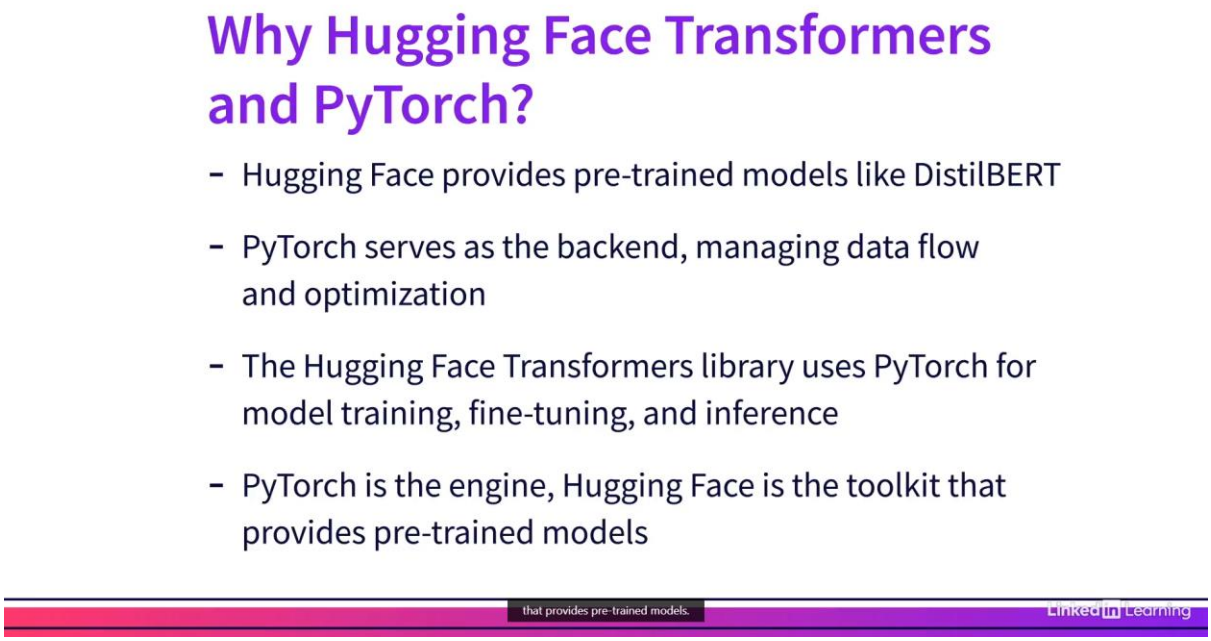
- PyTorch is a powerful open-source deep learning framework
- Widely used in both research and production environments
- Great performance and easy to use

It delivers great performance while being easy to use.

In This Course . . .

- We'll use PyTorch to build and train our neural network, and handle all the underlying computations
- PyTorch integrates seamlessly with the Hugging Face Transformers library

with Hugging Face Transformers Library.

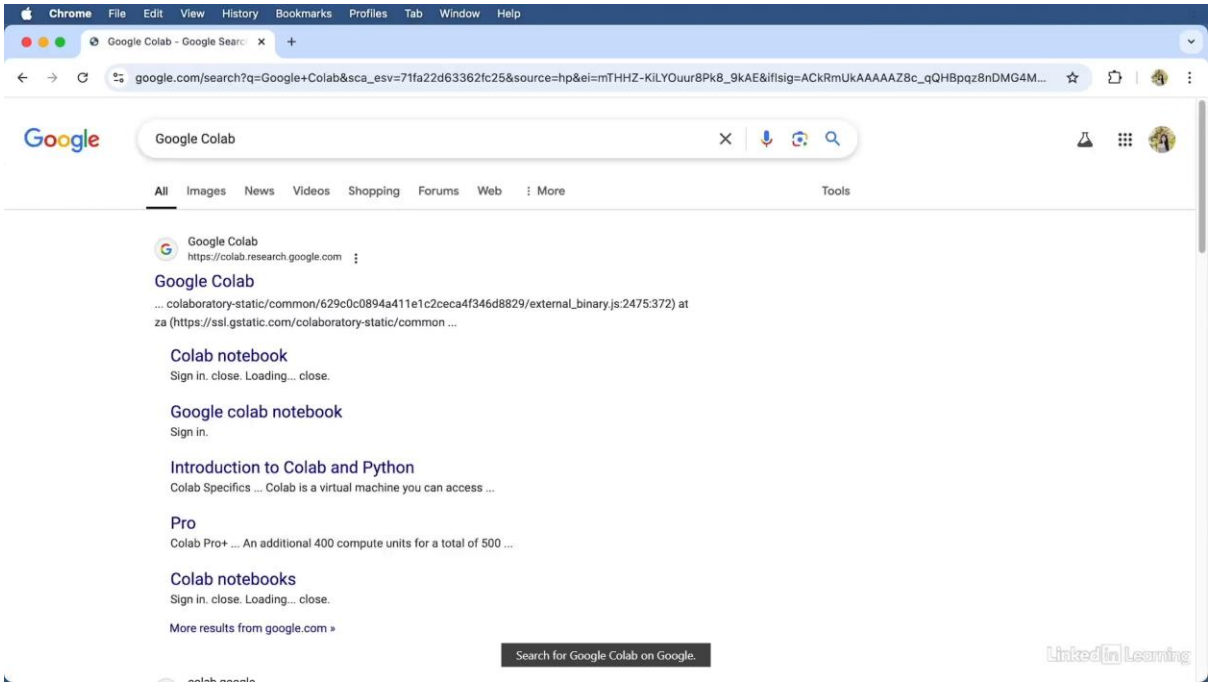


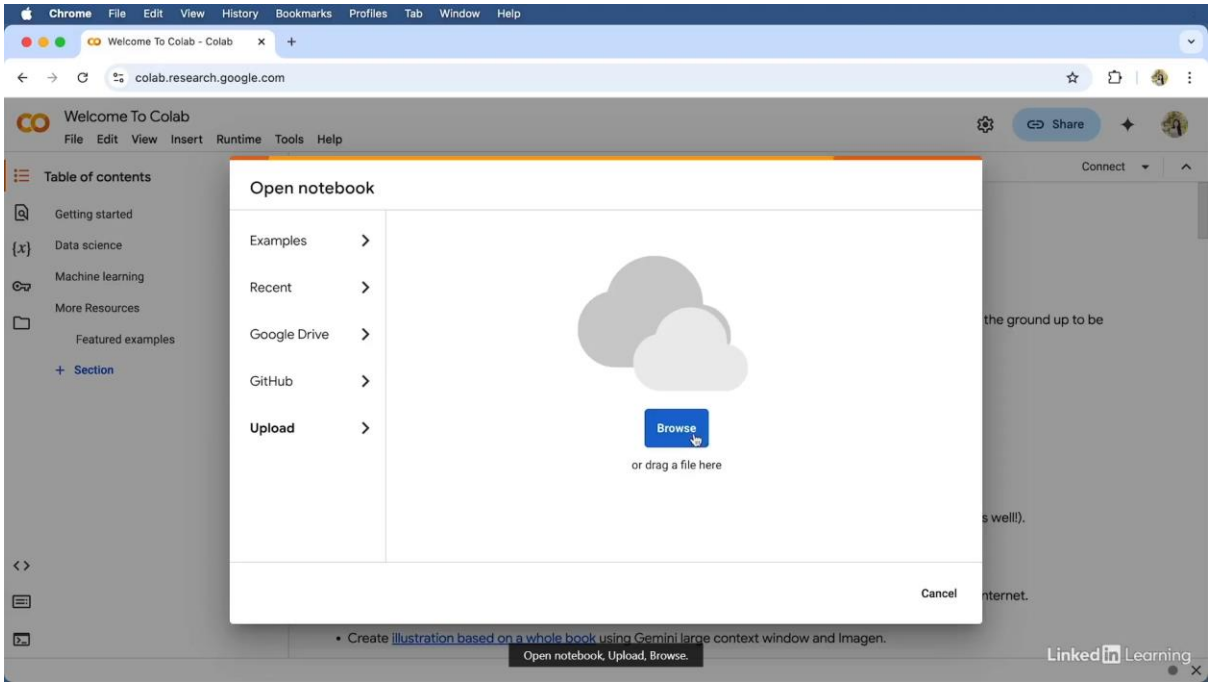
Setting up the environment on Google Colab

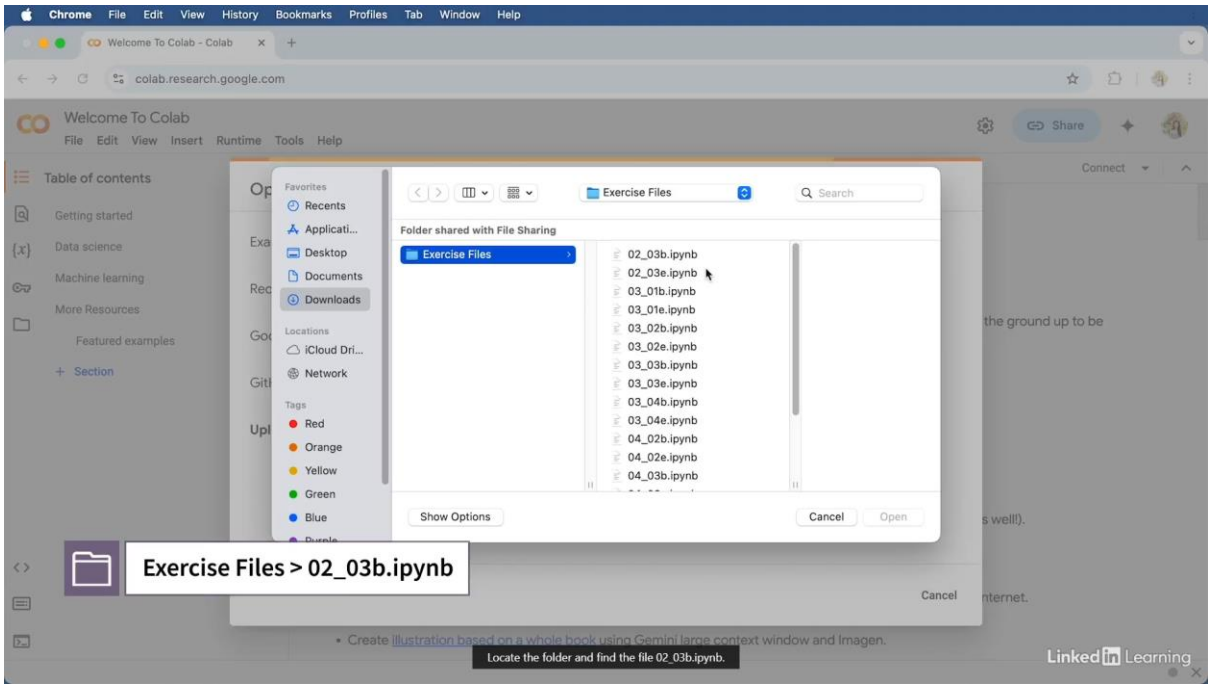
Selecting transcript lines in this section will navigate to timestamp in the video

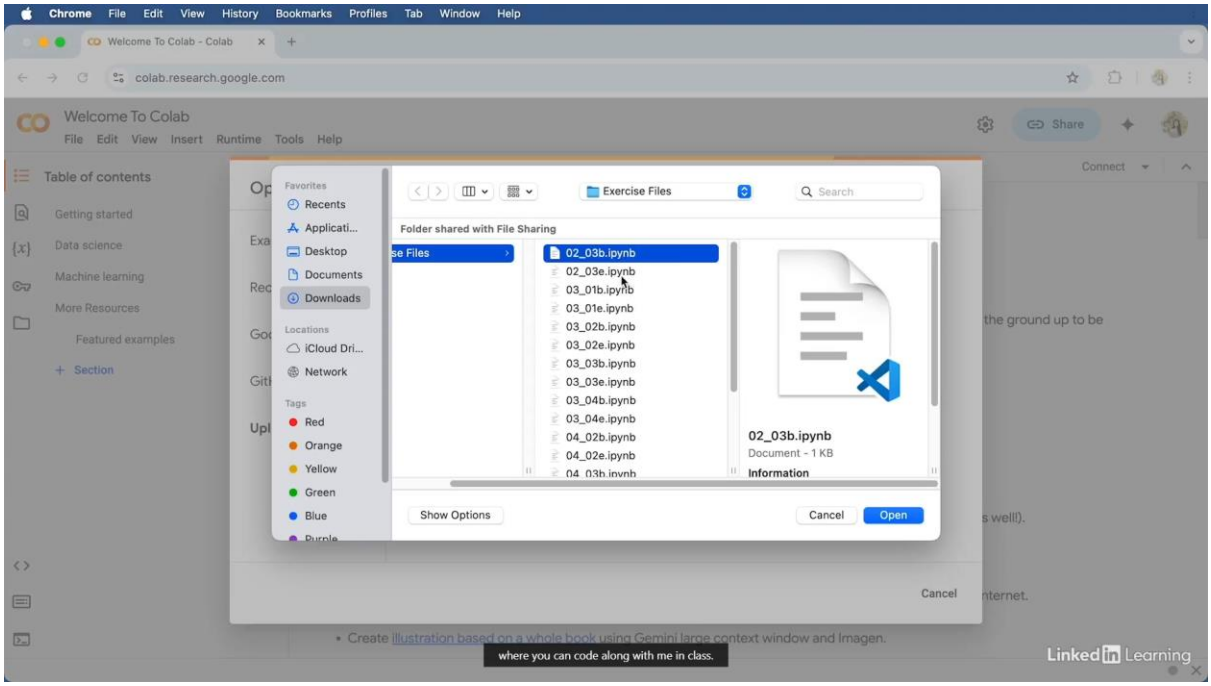
- [Instructor] We will be coding on a free cloud-based platform called Google Colab. It lets you run Python code without installing anything on your PC. But first, please download the Exercise Files folder from the Overview section of

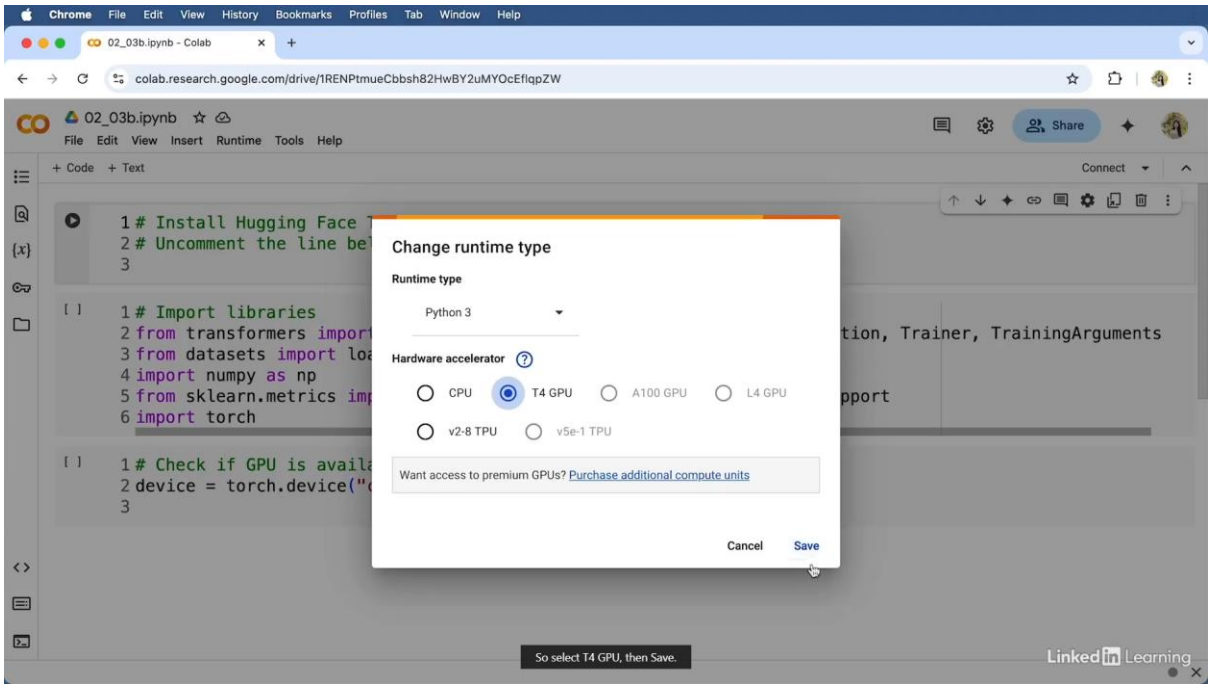
this course. Then make sure you're logged into your Google account. Search for Google Colab on Google. Click the first link, Cancel, and go to File, Open notebook, Upload, Browse. Locate the folder and find the file 02_03b.ipynb. The letter b in the file name indicates it is the beginning state of the file where you can code along with me in class. The files with letter e are the completed code for your reference after the class. In later videos, you'll find instructions on which Exercise File to use. Now let's open the file for this video. Before we start coding, let's click on Runtime, Change runtime type. We can see the current hardware accelerator is CPU, but Google Colab offers free GPU support, which is much stronger. So select T4 GPU, then Save. Now let's install three libraries. In the first cell, type the following code and hit Run button right here to install Hugging Face Transformers, datasets, and PyTorch libraries. You can also press Shift and Enter to run a code faster. Next, we'll import libraries for dataset loading, tokenization, model building, evaluation matrix, and so on. We will give them more introduction later in this course. Finally, let's confirm that we have access to GPU. If the name of the device is cuda, then your all set. FYI, every time you run a cell, the code will be automatically saved. Great. With the environment ready, we can now proceed to load and pre-process the dataset.

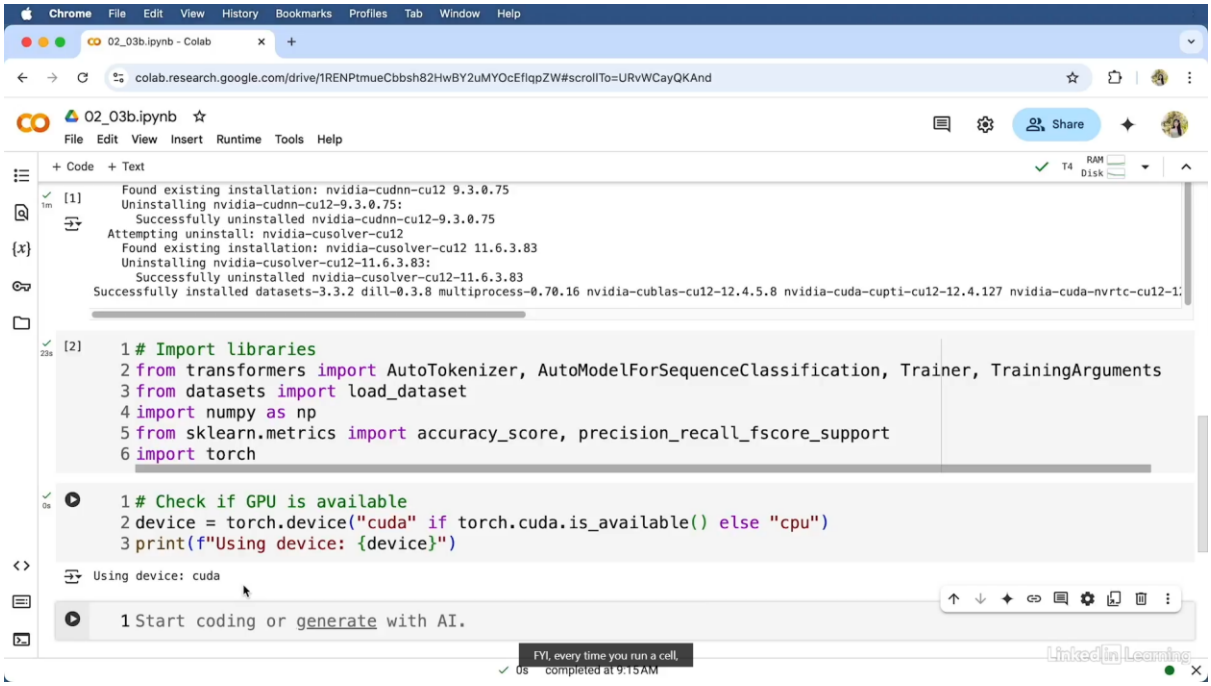


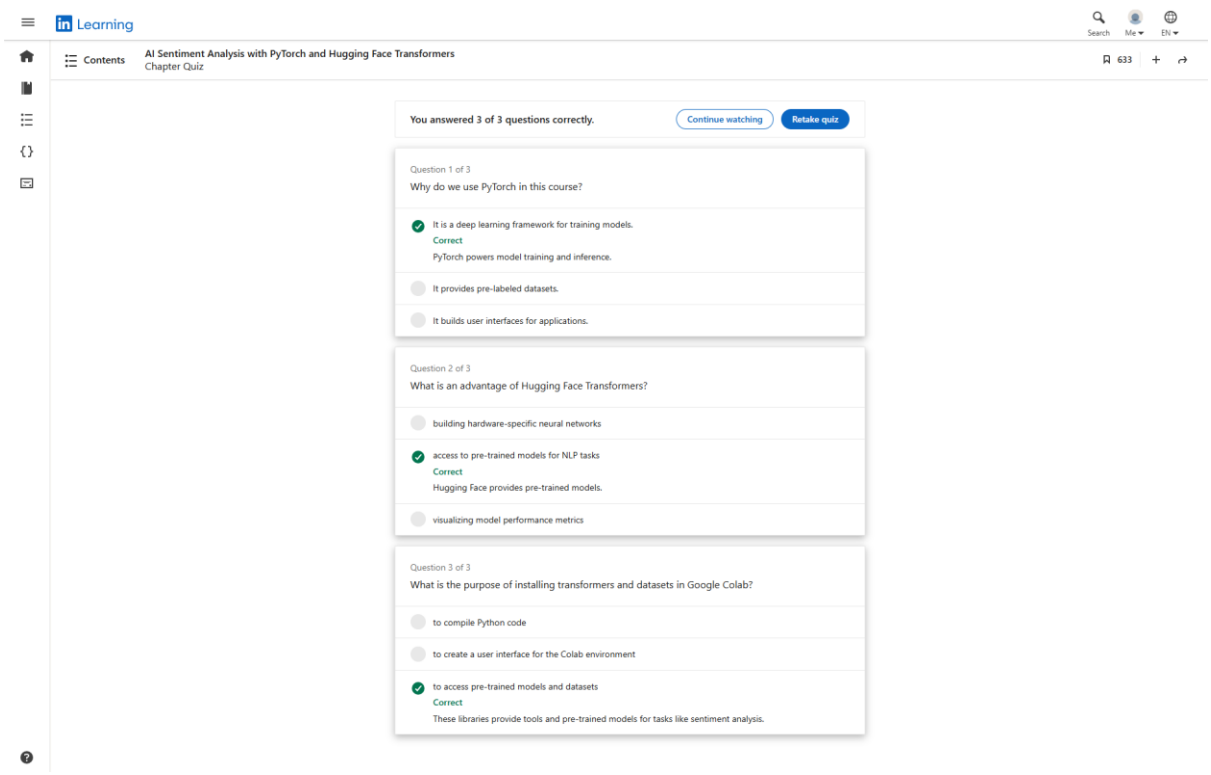

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Welcome To Colab - Colab
colab.research.google.com
Welcome To Colab
File Edit View Insert Runtime Tools Help
Share
Connect
Table of contents
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
Open notebook
Examples
Recent
Google Drive
GitHub
Upload
Browse
or drag a file here
Cancel
the ground up to be
s well!).
nternet.
• Create illustration based on a whole book using Gemini large context window and Imagen.
Open notebook, Upload, Browse.
LinkedIn Learning


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Welcome To Colab - Colab
colab.research.google.com
Welcome To Colab
File Edit View Insert Runtime Tools Help
Share
Connect
Table of contents
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
Favorites
Recents
Applicati...
Desktop
Documents
Downloads
Locations
iCloud Dri...
Network
Tags
Red
Orange
Yellow
Green
Blue
Exercise Files
Search
Folder shared with File Sharing
Exercise Files
02_03b.ipynb
02_03e.ipynb
03_01b.ipynb
03_01e.ipynb
03_02b.ipynb
03_02e.ipynb
03_03b.ipynb
03_03e.ipynb
03_04b.ipynb
03_04e.ipynb
04_02b.ipynb
04_02e.ipynb
04_03b.ipynb
Show Options
Cancel
Open
Exercise Files > 02_03b.ipynb
Cancel
the ground up to be
s well!).
nternet.
• Create illustration based on a whole book using Gemini large context window and Imagen.
Locate the folder and find the file 02_03b.ipynb.
LinkedIn Learning


Welcome To Colab
Table of contents
Exercise Files
Folder shared with File Sharing
02_03b.ipynb
Document - 1 KB
Information
Show Options
Cancel
Open
where you can code along with me in class.
LinkedIn Learning


02_03b.ipynb - Colab
colab.research.google.com/drive/1RENPtmueCbbsh82HwBY2uMYOcEflqpZW
Change runtime type
Runtime type
Python 3
Hardware accelerator
CPU
T4 GPU
A100 GPU
L4 GPU
v2-8 TPU
v5e-1 TPU
Want access to premium GPUs? Purchase additional compute units
Cancel
Save
So select T4 GPU, then Save.
LinkedIn Learning

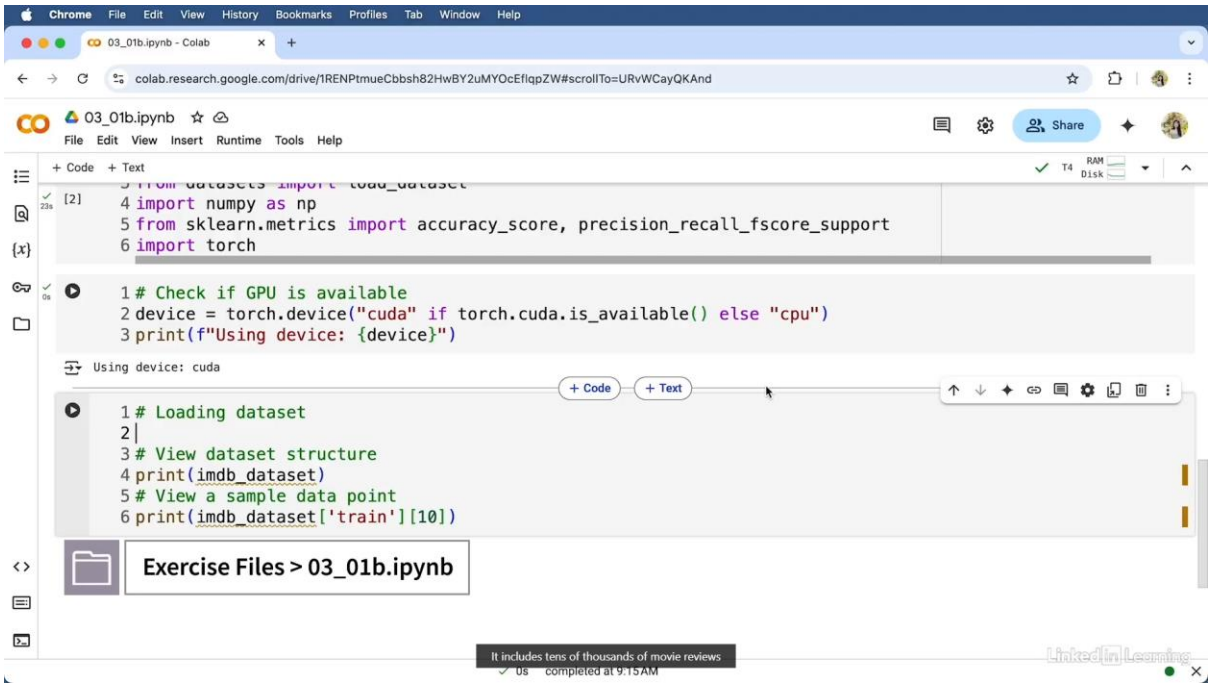


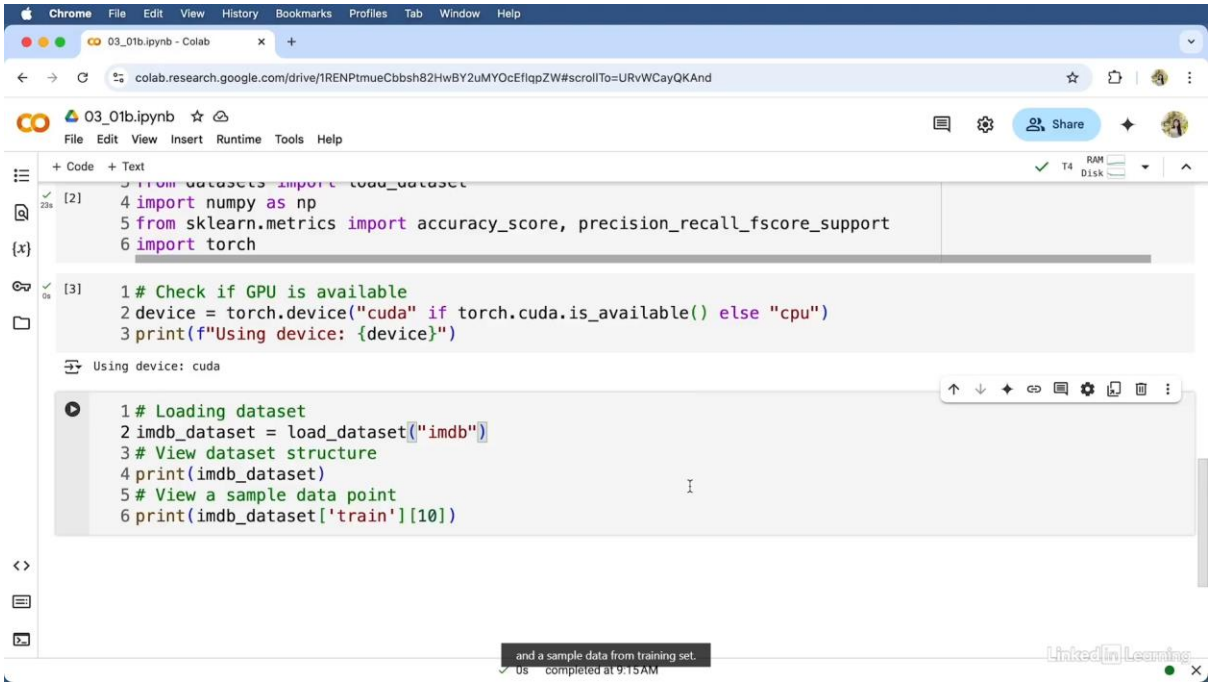


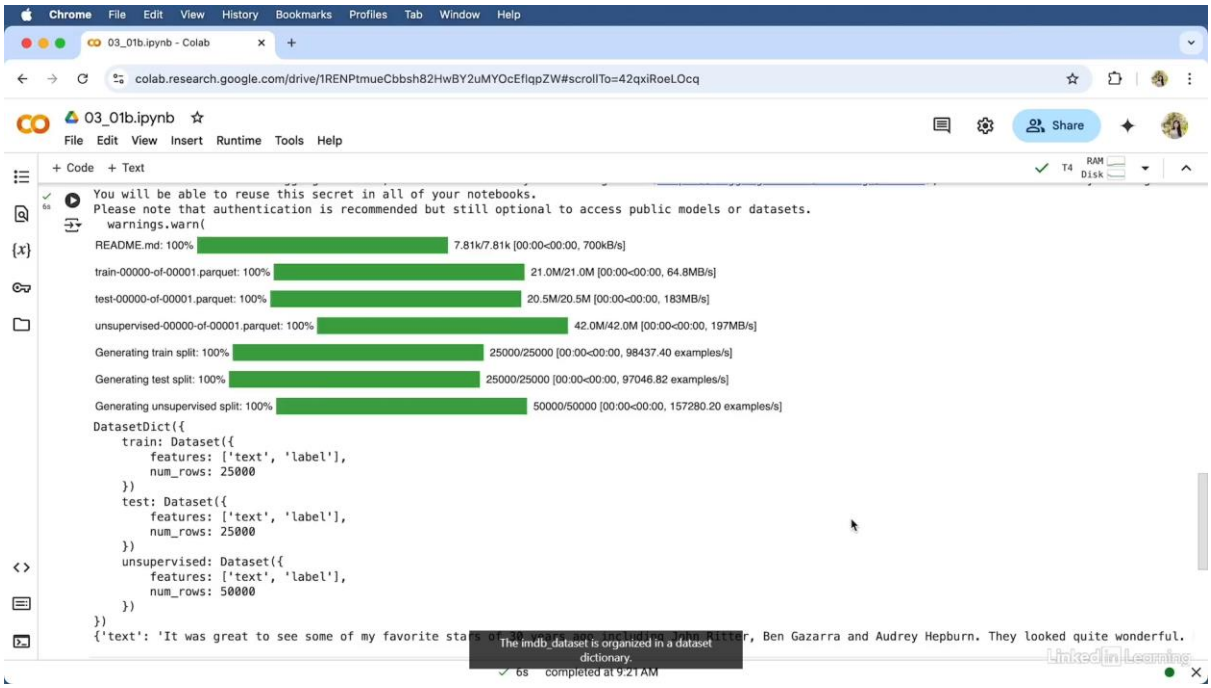
Loading dataset

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The dataset we are going to use in this course is the imdb_dataset. It includes tens of thousands of movie reviews with their sentiment labels. We load the imdb_dataset with Hugging Face datasets library. The function is called load_dataset, which is already imported. (keyboard clicking) Once we load the dataset, we can print the dataset structure and a sample data from training set. The imdb_dataset is organized in a dataset dictionary. There are three subdatasets, the training set, test set, and unsupervised set. We can ignore the third one for now. The features show that each set contains movie reviews and their sentiment labels. The training set is used to train our model, While the test set is used to evaluate model performance. The training and test sets each contain 25,000 samples. For the sample data point, we can see that each data item contains a movie review and a sentiment label where zero represents negative and one represents positive.





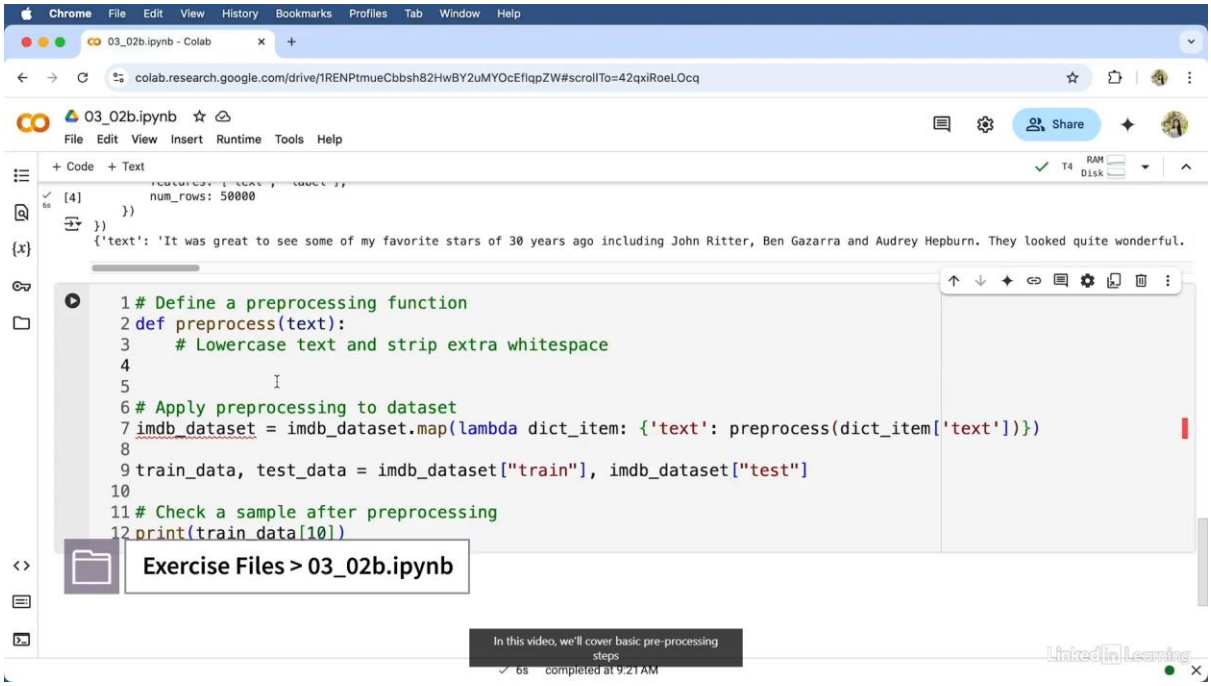


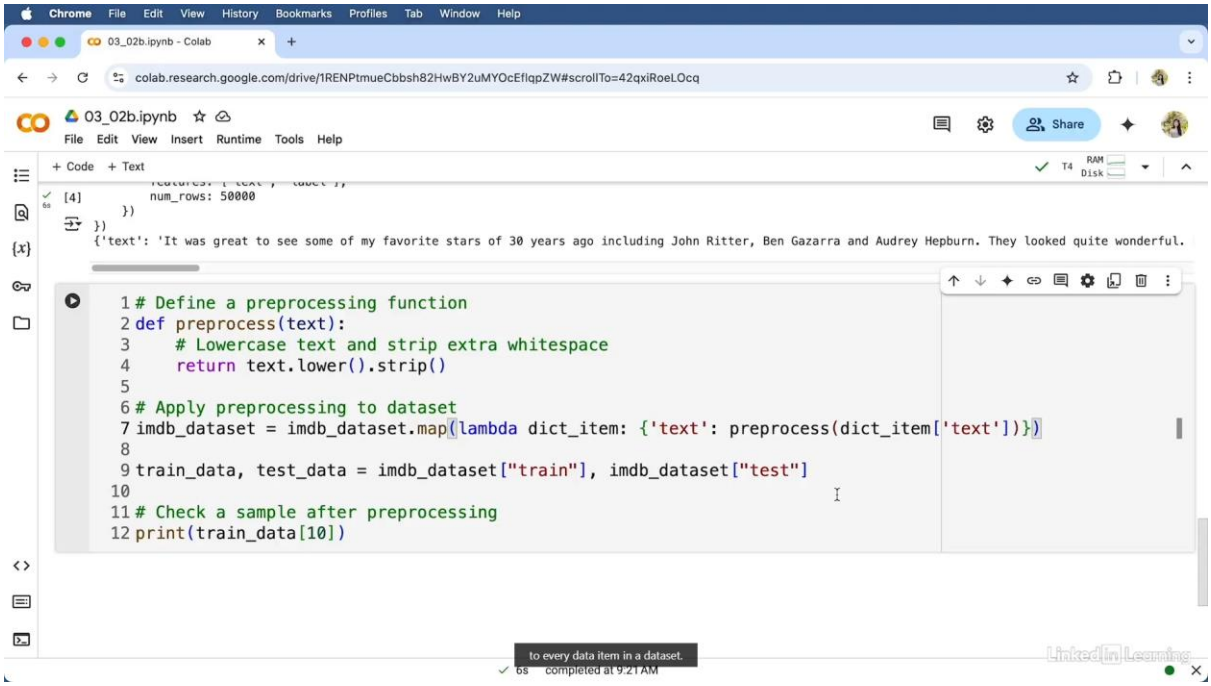
Preprocessing text data

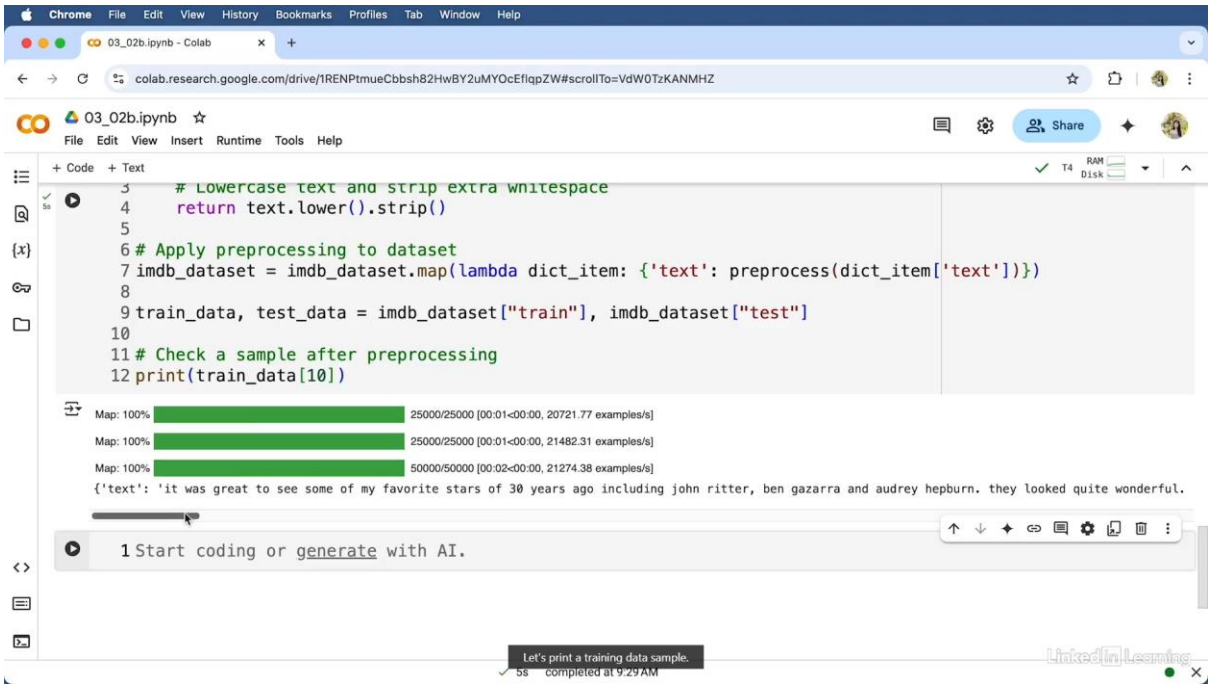
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Data pre-processing is a crucial step in natural language processing. In this video, we'll cover basic pre-processing steps like converting text to lowercase and removing extra white space. We first define

a pre-processing function. This function will take a piece of text, lowercase it, and remove any leading or trailing white space. The lower function ensures the model treats words with the same spelling, but different capitalizations as identical. The strip function removes extra white space that could affect the tokenization process. The dataset's library from Hugging Face provides a map function that applies the pre-processing function to every data item in a dataset. We also split the pre-processed dataset into training and test sets. Finally, we'll verify if our pre-processing function is doing its job. Let's print a training data sample. We can see the text is properly lowercased and extra white space has been removed. Excellent.





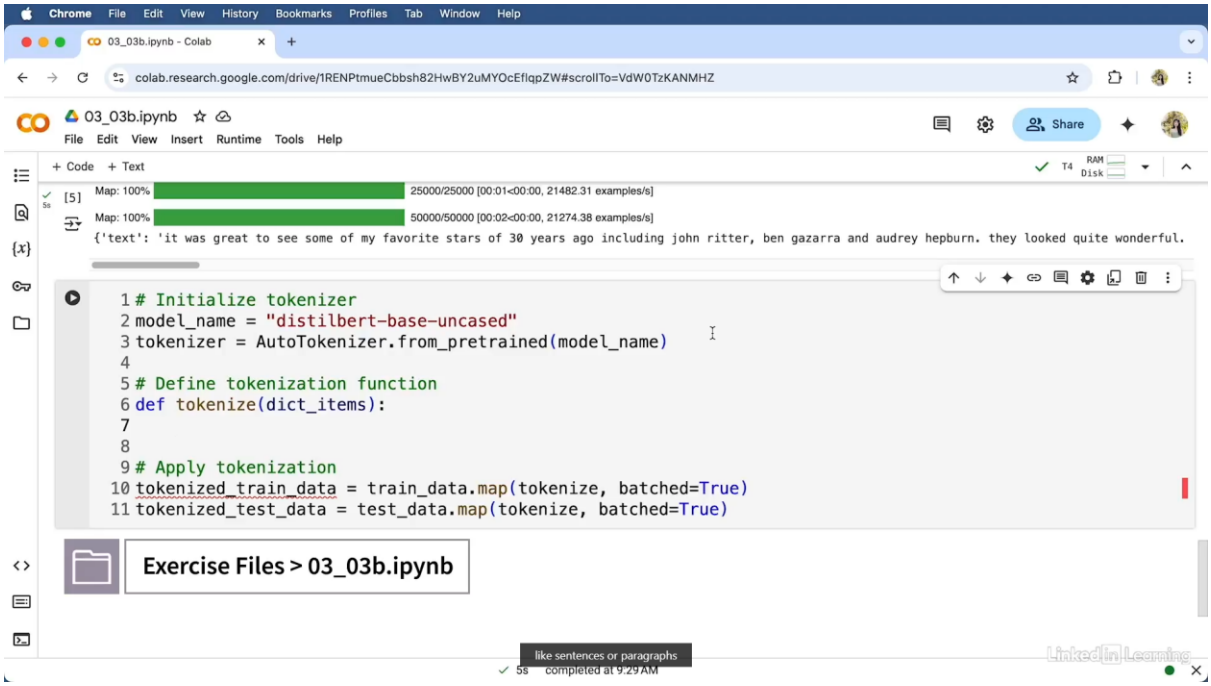


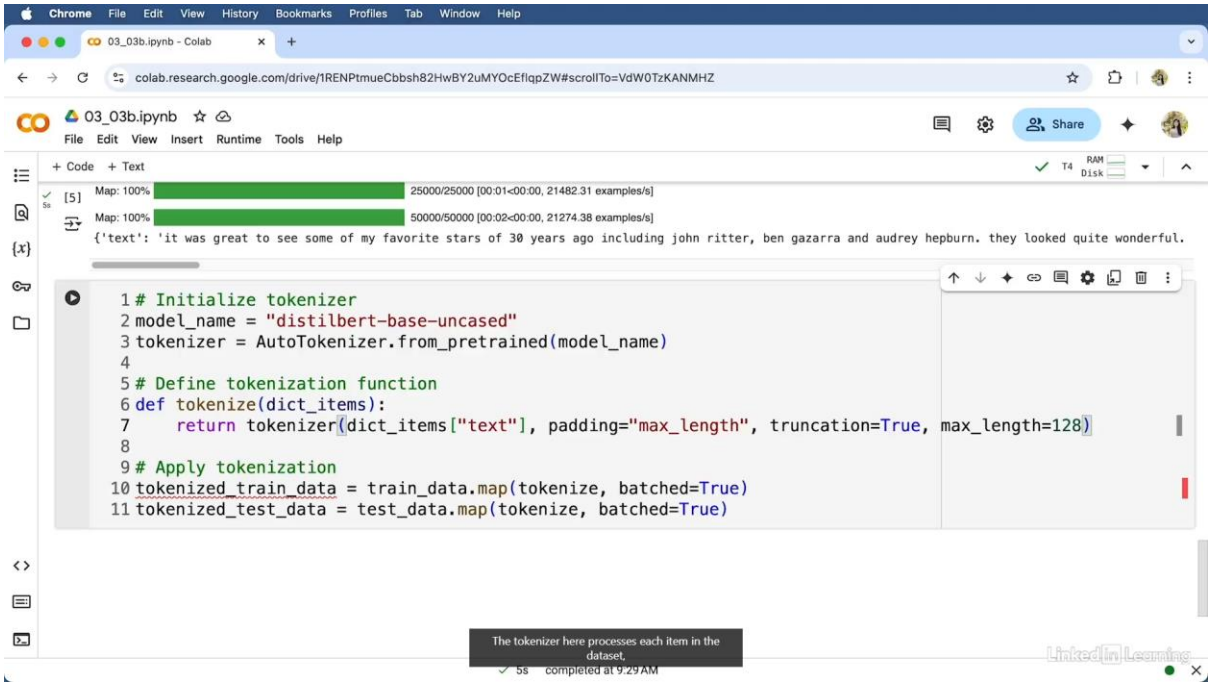
Tokenization

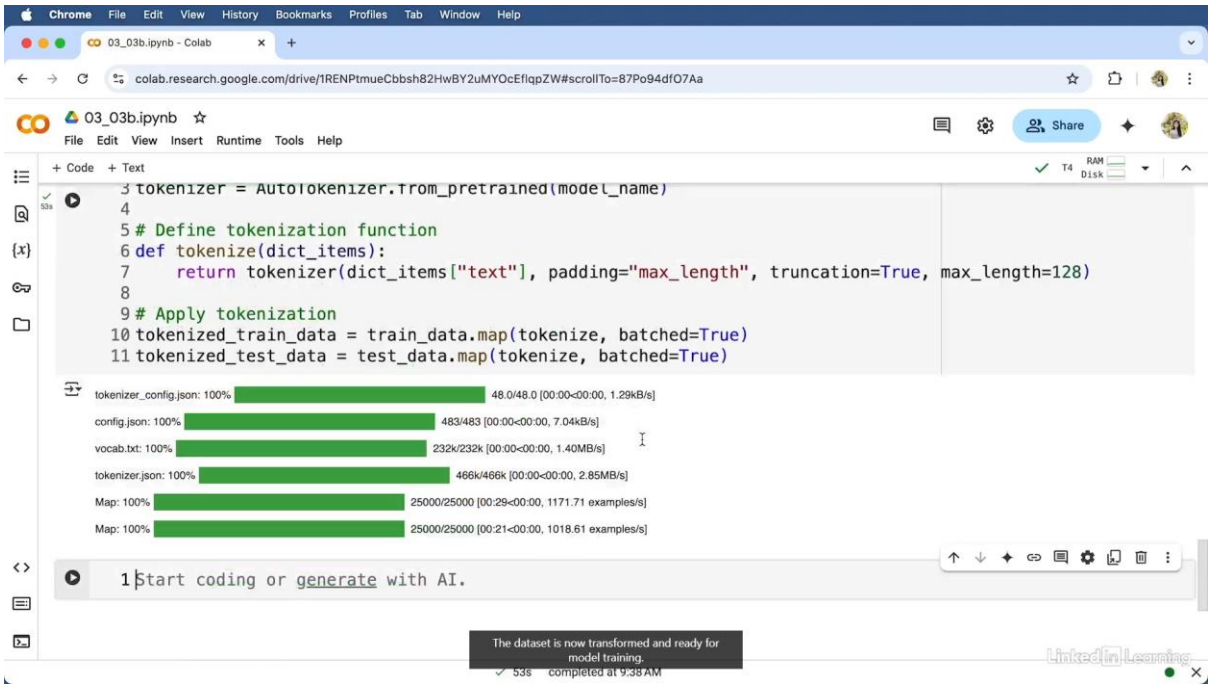
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Tokenization is the process of breaking down larger pieces of text, like sentences or paragraphs into smaller units called tokens. For example, the sentence "I love movies," is split into tokens, I, love, and

movies. To create a tokenizer, we use AutoTokenizer class from Hugging Face. It initializes the tokenizer with the distilbert model name distilbert-base-uncased. Now, let's define a tokenization function with the tokenizer. (keyboard clicking) The tokenizer here processes each item in the dataset, the parameters for padding, truncation, and max_length, make sure all sequences are of the same length, and are less than 128 tokens. These steps are necessary because our model expects a fixed-size input. Now, we apply the tokenized function to both the training and test datasets using the map function from datasets library. This tokenizes the data efficiently in batches. The dataset is now transformed and ready for model training. In the next video, we'll introduce padding and truncation with an example.





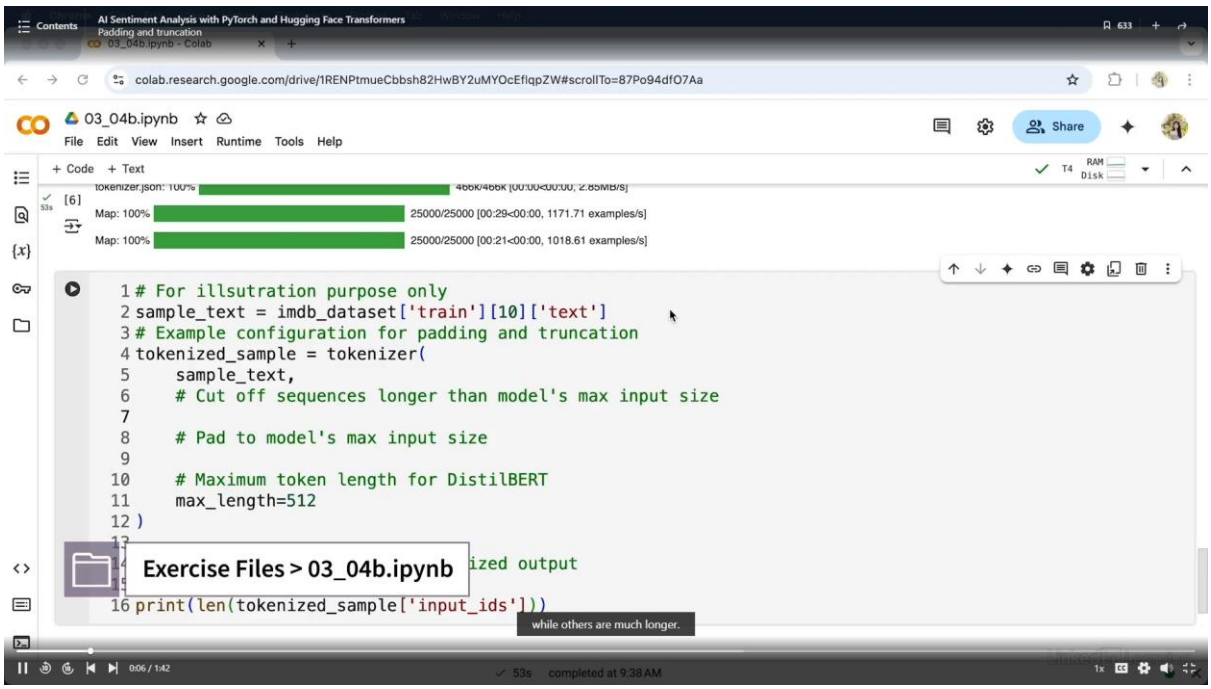


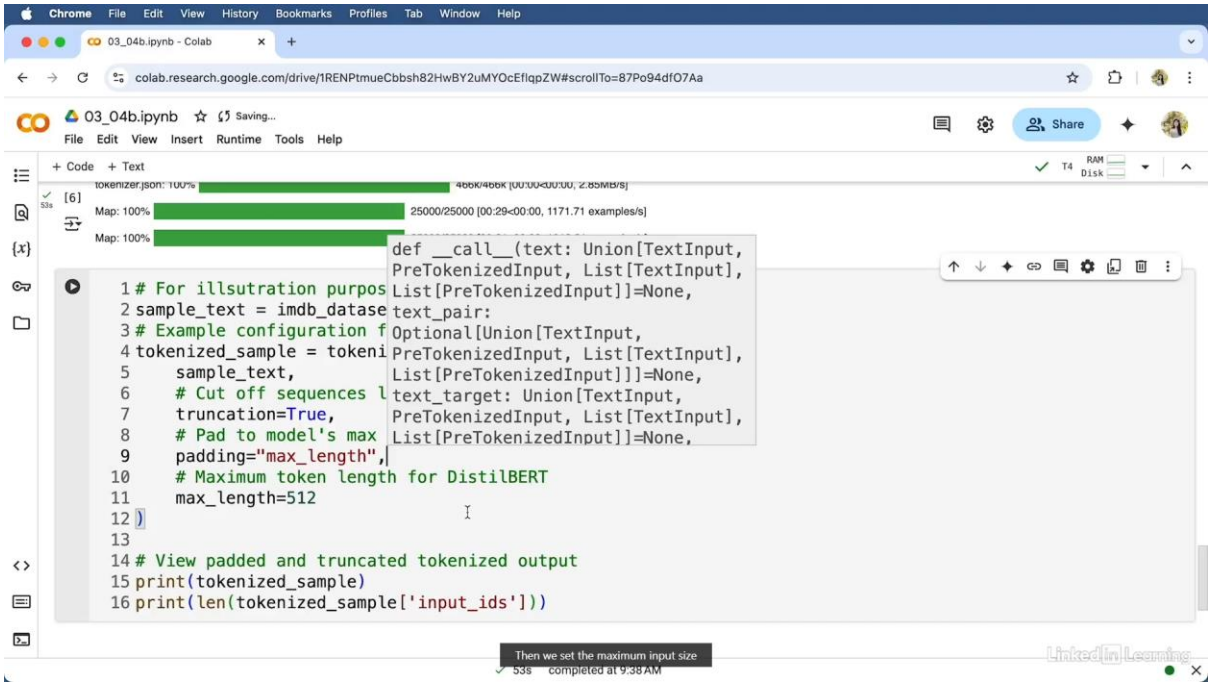
Padding and truncation

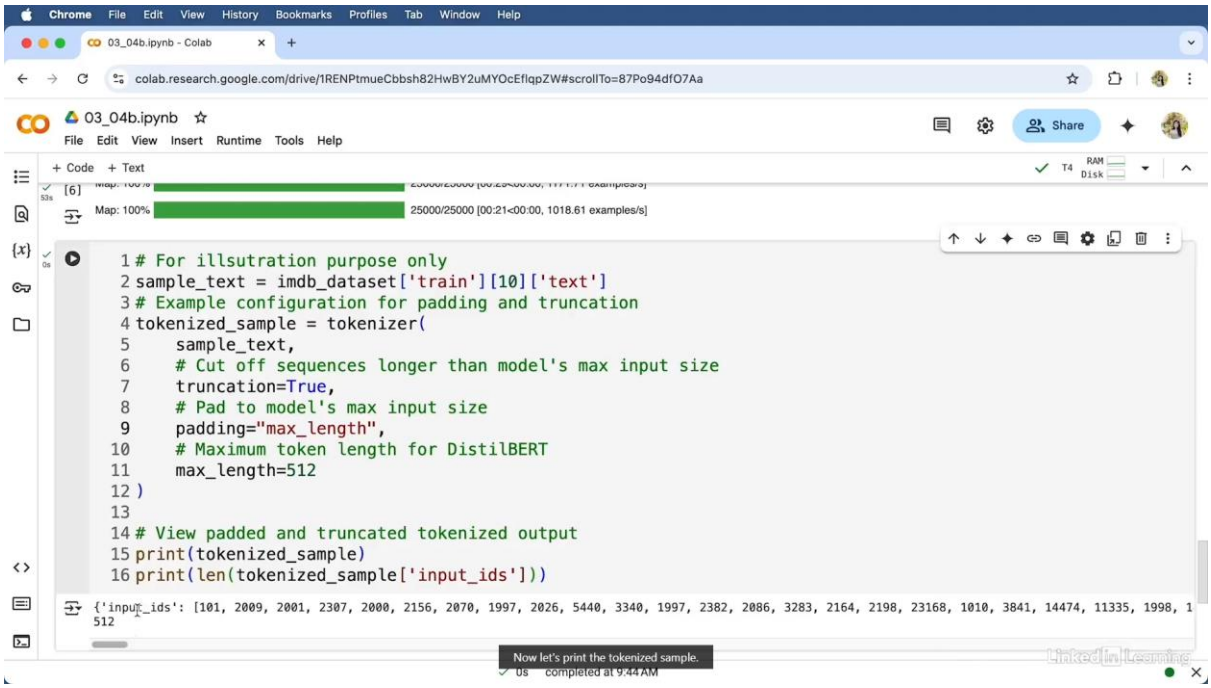
Selecting transcript lines in this section will navigate to timestamp in the video

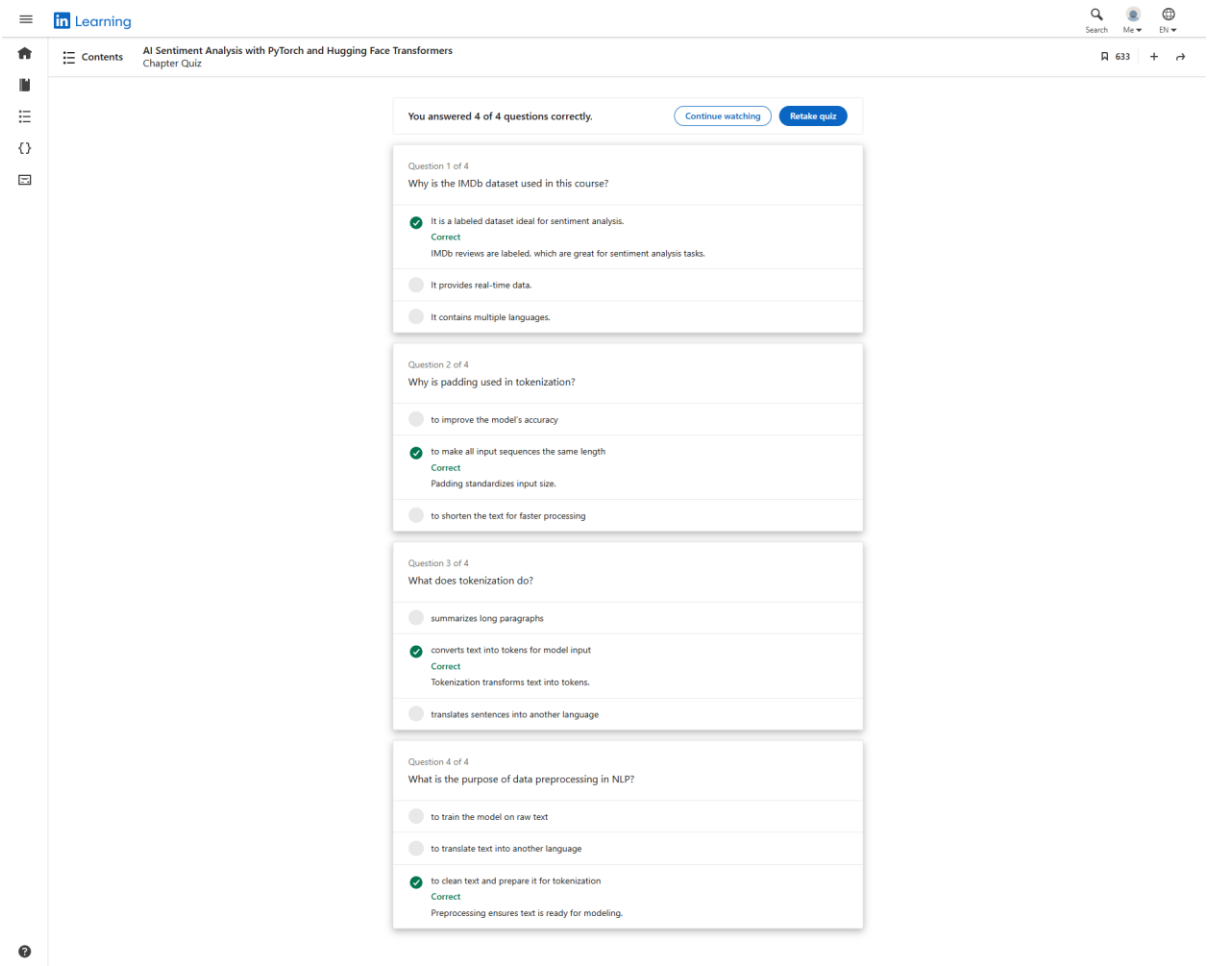
- [Instructor] You may have noticed text data varies in length. Some sentences are short, while others are much longer. Models like DistilBERT usually require inputs of the same length, which is where padding and

truncation come into play. Padding ensures all sequences have the same length by adding extra tokens, often zeros, to shorter sentences. On the other hand, truncation shortens longer sequences to a specified length. Both steps standardize the input size for the model. The tokenizer we used earlier automatically handles padding and truncation. To demonstrate how this works, we'll apply them to another sample from the dataset. First, we tokenize the sample text, while specifying padding and truncation parameters. We set the truncation to true and padding to max length. Then we set the maximum input size to 512 tokens. Now let's print the tokenized sample. The input IDs here are the numerical representation of the tokenized sequence. We also print the length of the sample to confirm it satisfies the maximum length requirement. It looks like padding and truncation are working as expected.




03_04b.ipynb - Colab
colab.research.google.com/drive/1RENPtmueCbbsh82HwBY2uMYOcEflqpZW#scrollTo=87Po94dfO7Aa
03_04b.ipynb
Share
+ Code + Text
Map: 100% 25000/25000 [00:29<00:00, 1171.71 examples/s]
Map: 100%
def __call__(text: Union[TextInput, PreTokenizedInput, List[TextInput], List[PreTokenizedInput]]=None, text_pair: Optional[Union[TextInput, PreTokenizedInput, List[TextInput], List[PreTokenizedInput]]]=None, text_target: Union[TextInput, PreTokenizedInput, List[TextInput], List[PreTokenizedInput]]=None,
# For illsutration purpos
sample_text = imdb_datase
# Example configuration f
tokenized_sample = tokeni
sample_text,
# Cut off sequences l
truncation=True,
# Pad to model's max
padding="max_length",
# Maximum token length for DistilBERT
max_length=512
)
# View padded and truncated tokenized output
print(tokenized_sample)
print(len(tokenized_sample['input_ids']))
Then we set the maximum input size
53s completed at 9:38 AM


03_04b.ipynb - Colab
colab.research.google.com/drive/1RENPtmueCbbsh82HwBY2uMYOcEflqpZW#scrollTo=87Po94dfO7Aa
03_04b.ipynb
Share
+ Code + Text
Map: 100% 25000/25000 [00:21<00:00, 1018.61 examples/s]
# For illsutration purpose only
sample_text = imdb_dataset['train'][10]['text']
# Example configuration for padding and truncation
tokenized_sample = tokenizer(
sample_text,
# Cut off sequences longer than model's max input size
truncation=True,
# Pad to model's max input size
padding="max_length",
# Maximum token length for DistilBERT
max_length=512
)
# View padded and truncated tokenized output
print(tokenized_sample)
print(len(tokenized_sample['input_ids']))
{'input_ids': [101, 2009, 2001, 2307, 2000, 2156, 2070, 1997, 2026, 5440, 3340, 1997, 2382, 2086, 3283, 2164, 2198, 23168, 1010, 3841, 14474, 11335, 1998, 1
512
Now let's print the tokenized sample.
0s completed at 9:44 AM

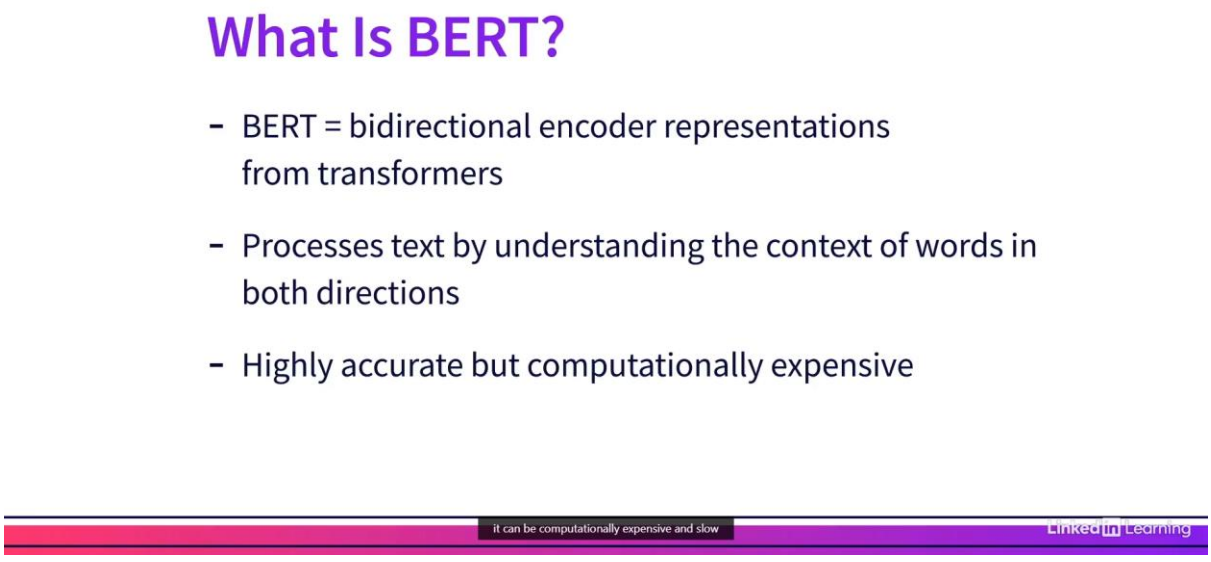


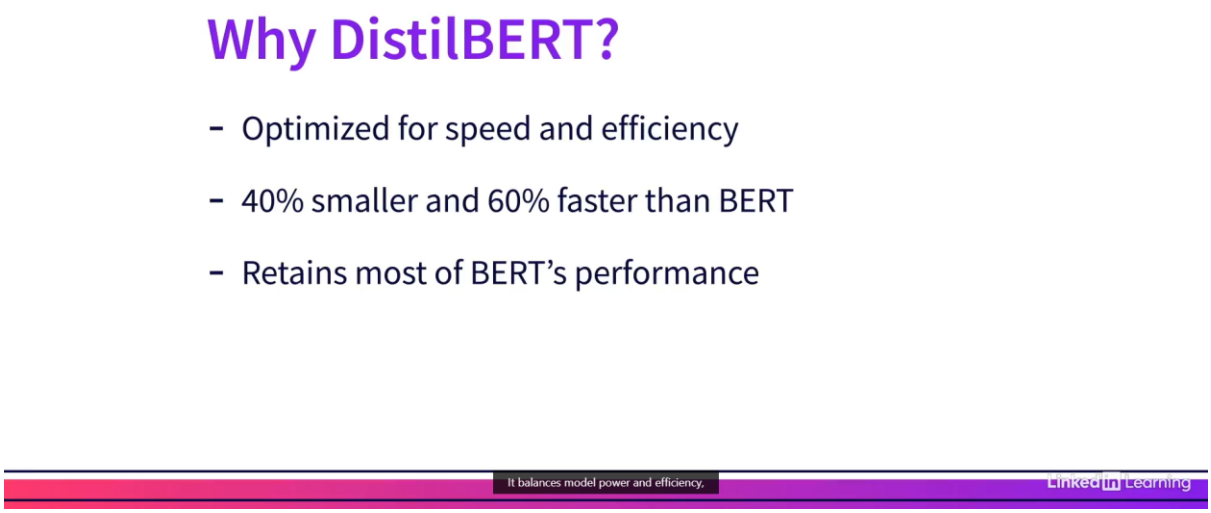
What is DistilBERT?

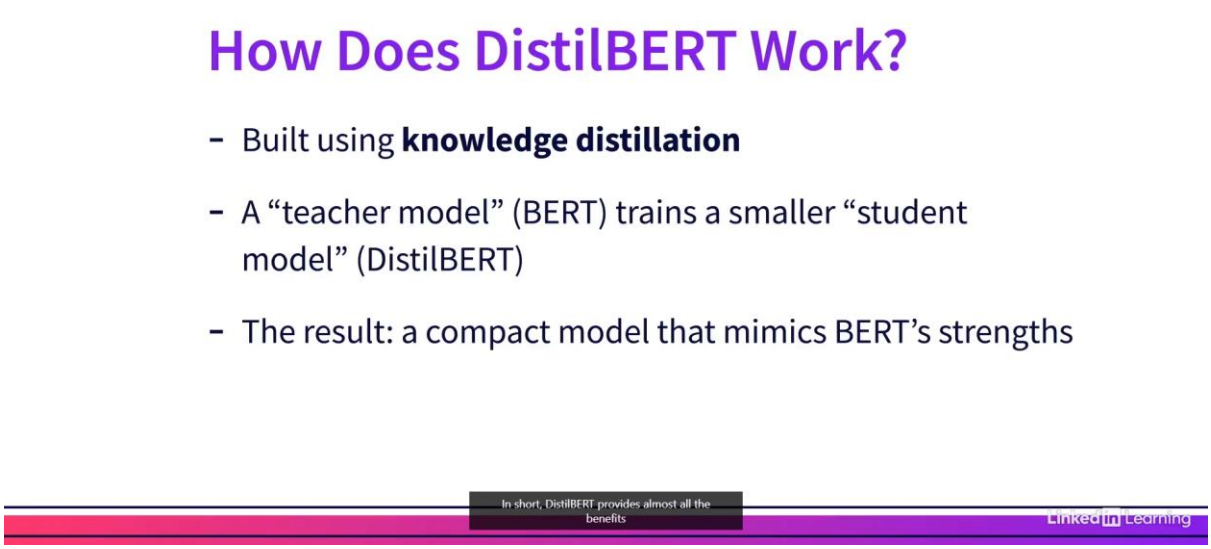
Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Imagine a model that's nearly as powerful as BERT, but smaller and faster. That's exactly what DistilBERT offers. In this video, we'll explore what makes DistilBERT unique and why it's ideal for our project. But first, what is BERT? BERT stands for bi-directional encoder representations from transformers. It processes text by understanding the context of words in both directions. For example, in a sentence, "The bank by the river," BERT recognizes that bank refers to a river bank, not a financial institution by analyzing the surrounding context. While BERT is very powerful, it can be computationally expensive and slow for real world applications. DistilBERT is 40% smaller and 60% faster while keeping most of BERT's performance. It balances model power and efficiency, even when you're processing large

scale data. DistilBERT involves a process called knowledge distillation. Here's how it works. A larger model called the teacher model, in this case BERT, trains a smaller model, known as the student model. The student learns to mimic the teacher's predictions and retain most of its capabilities. In short, DistilBERT provides almost all the benefits of BERT while being faster and lighter.





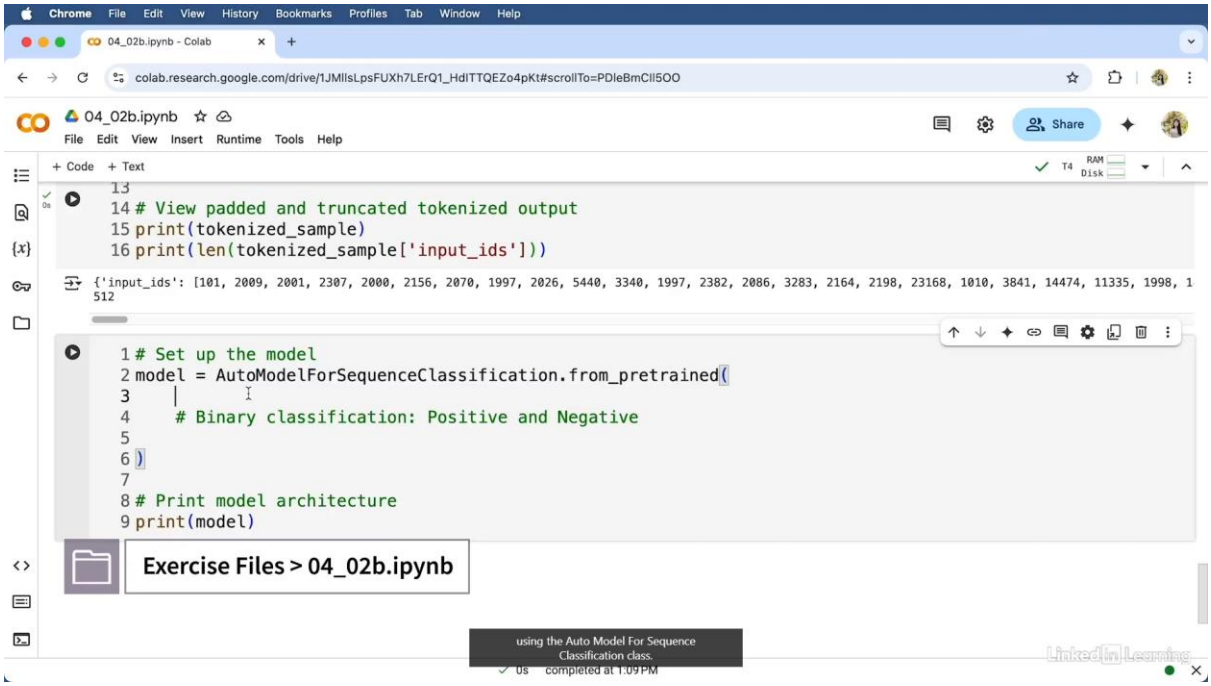


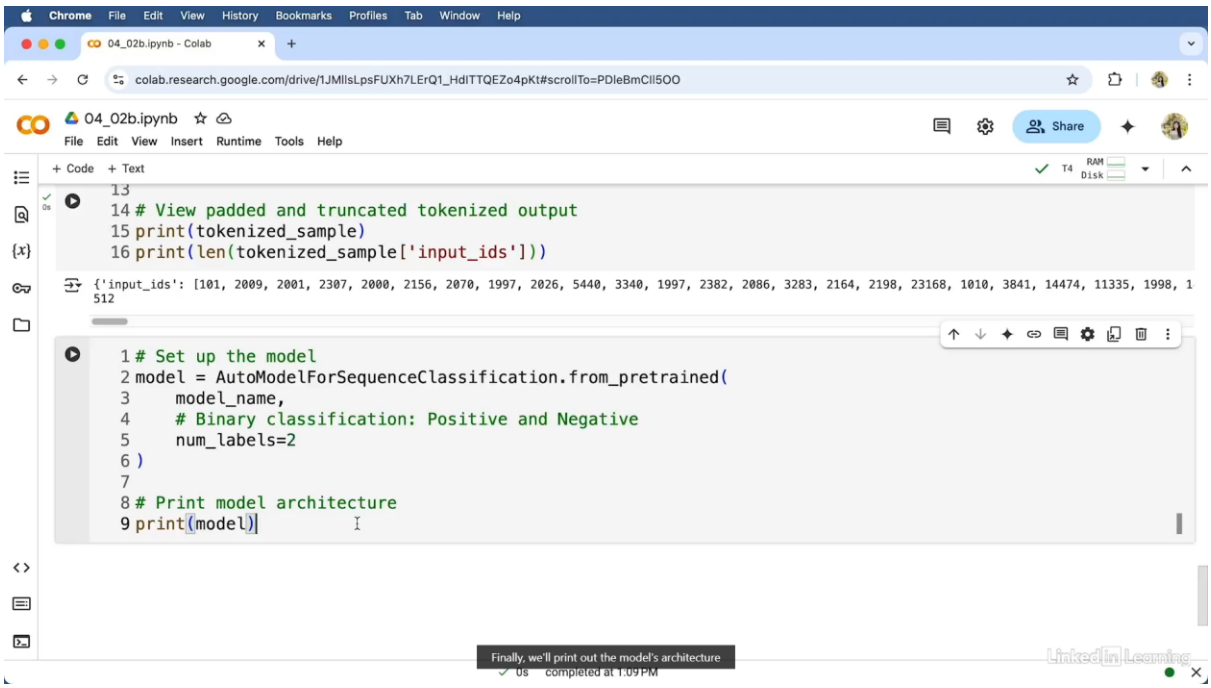
Setting up the model

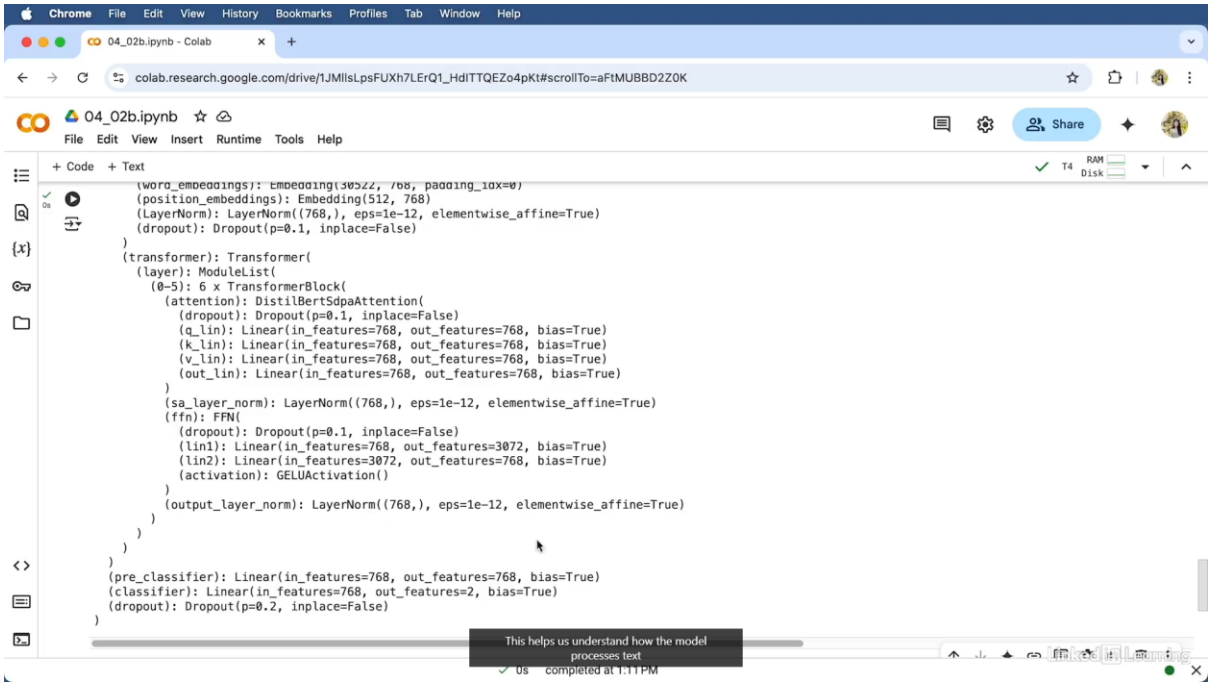
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video we'll set up the SteelBERT using the Auto Model For Sequence Classification class. The best part is we don't have to build the model from scratch. This pre-trained model has done all the heavy lifting for

us. Since sentiment analysis in our case is a binary classification problem, we pass in model name and set the number of labels to two, one for positive and one for negative. Finally, we'll print out the model's architecture to see the individual layers and components. This helps us understand how the model processes text and makes predictions.



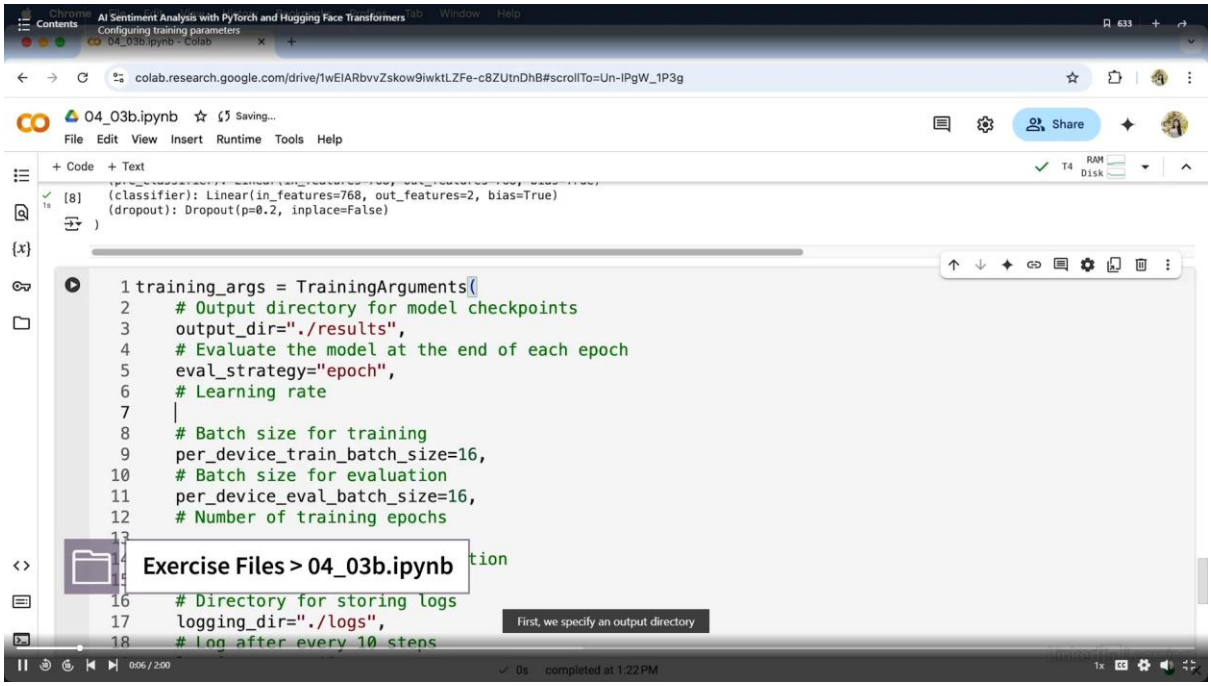


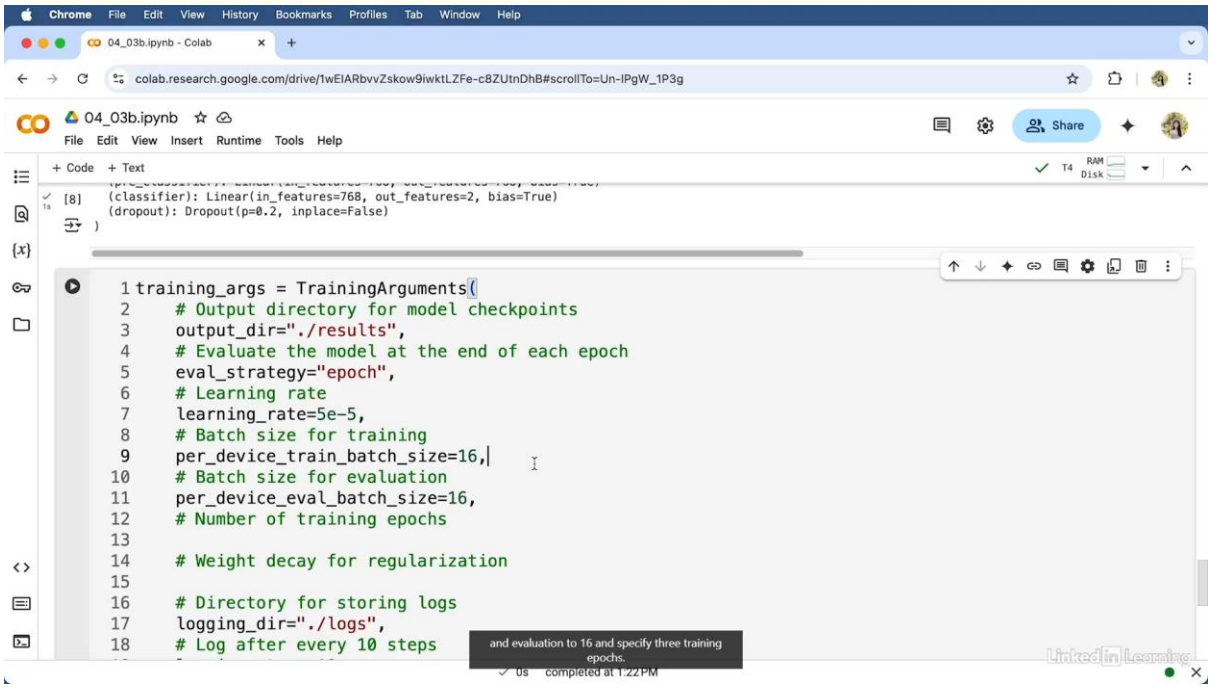


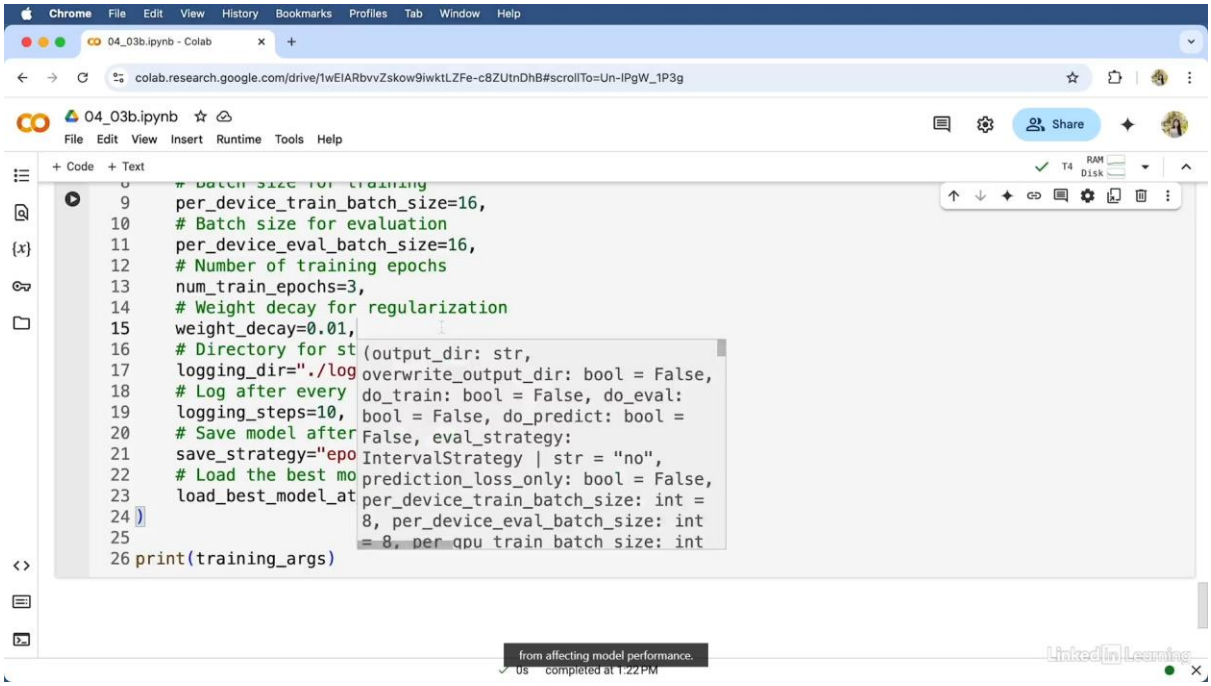
Configuring training parameters

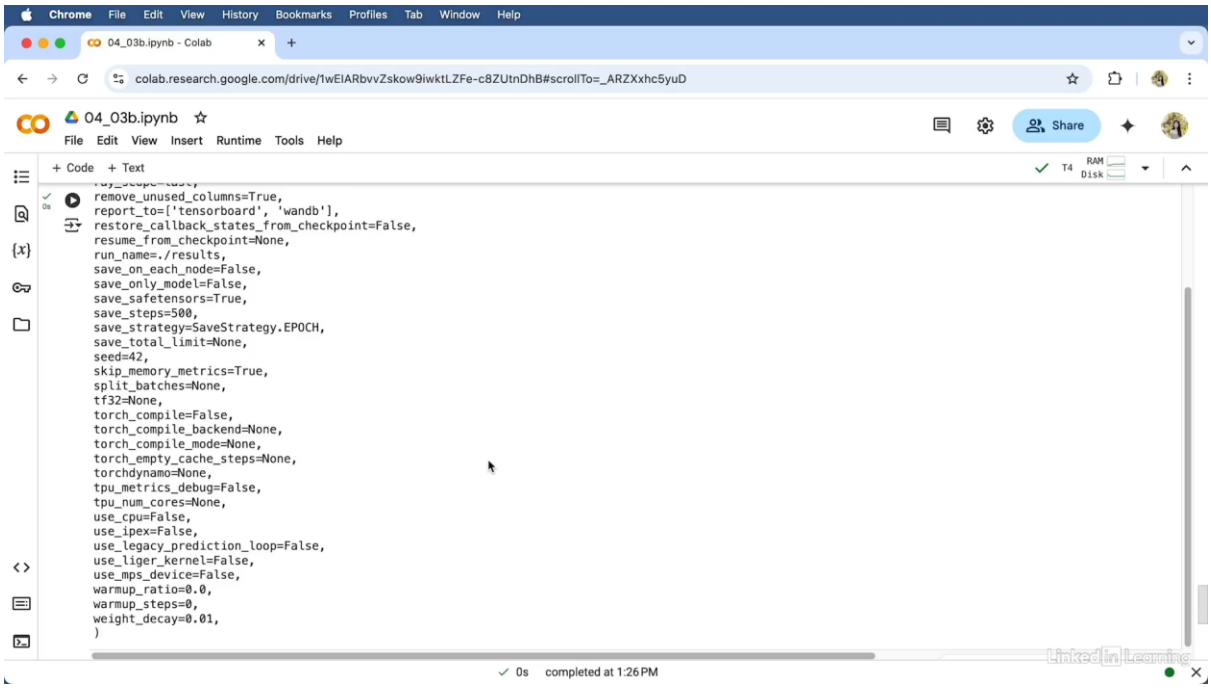
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We'll use the training arguments class from Hugging Face Transformers library. First, we specify an output directory to save model checkpoints. This helps track different training stages, so we can load the best model later. The evaluation strategy is set to epoch. This means the model will be evaluated at the end of each training epoch. The learning rate controls how much the model updates its parameters during training. We use a learning rate of 5e-5 which is a common choice for fine tuning pre-trained models. (keyboard clicking) We set the batch size for both training and evaluation to 16 and specify three training epochs. (keyboard clicking) Three, is a good starting point and we can adjust it based on the model's performance during training. To prevent over-fitting, we set weight decay to 0.01. This regularization technique prevents large weights from affecting model performance. We also log after every 10 steps and save the model after each epoch. Finally, we set the model to load its best version after training. Let's print the training arguments to confirm they're setup correctly. Everything looks good.


AI Sentiment Analysis with PyTorch and Hugging Face Transformers
Configuring training parameters
04_03b.ipynb - Colab
colab.research.google.com/drive/1wElARbvvZskow9iwktLZFe-c8ZUtnDhB#scrollTo=Un-IPgW_1P3g
04_03b.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
[8]
(classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.2, inplace=False)
)
training_args = TrainingArguments(
# Output directory for model checkpoints
output_dir="./results",
# Evaluate the model at the end of each epoch
eval_strategy="epoch",
# Learning rate
# Batch size for training
per_device_train_batch_size=16,
# Batch size for evaluation
per_device_eval_batch_size=16,
# Number of training epochs
# Directory for storing logs
logging_dir="./logs",
# Log after every 10 steps
Exercise Files > 04_03b.ipynb
First, we specify an output directory
0:06 / 2:00
0s completed at 1:22PM


Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_03b.ipynb - Colab
colab.research.google.com/drive/1wElARbvvZskow9iwktLZFe-c8ZUtnDhB#scrollTo=Un-IPgW_1P3g
04_03b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
[8]
(classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.2, inplace=False)
)
training_args = TrainingArguments(
# Output directory for model checkpoints
output_dir="./results",
# Evaluate the model at the end of each epoch
eval_strategy="epoch",
# Learning rate
learning_rate=5e-5,
# Batch size for training
per_device_train_batch_size=16,
# Batch size for evaluation
per_device_eval_batch_size=16,
# Number of training epochs
# Weight decay for regularization
# Directory for storing logs
logging_dir="./logs",
# Log after every 10 steps
and evaluation to 16 and specify three training epochs.
0s completed at 1:22PM



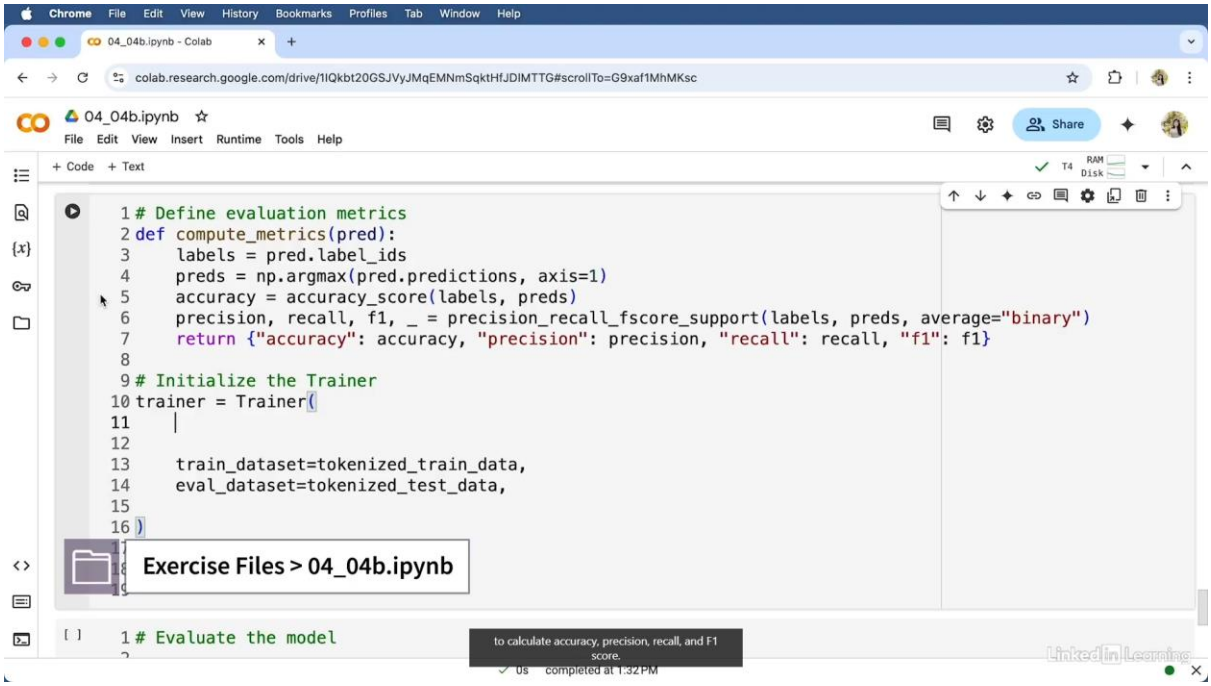


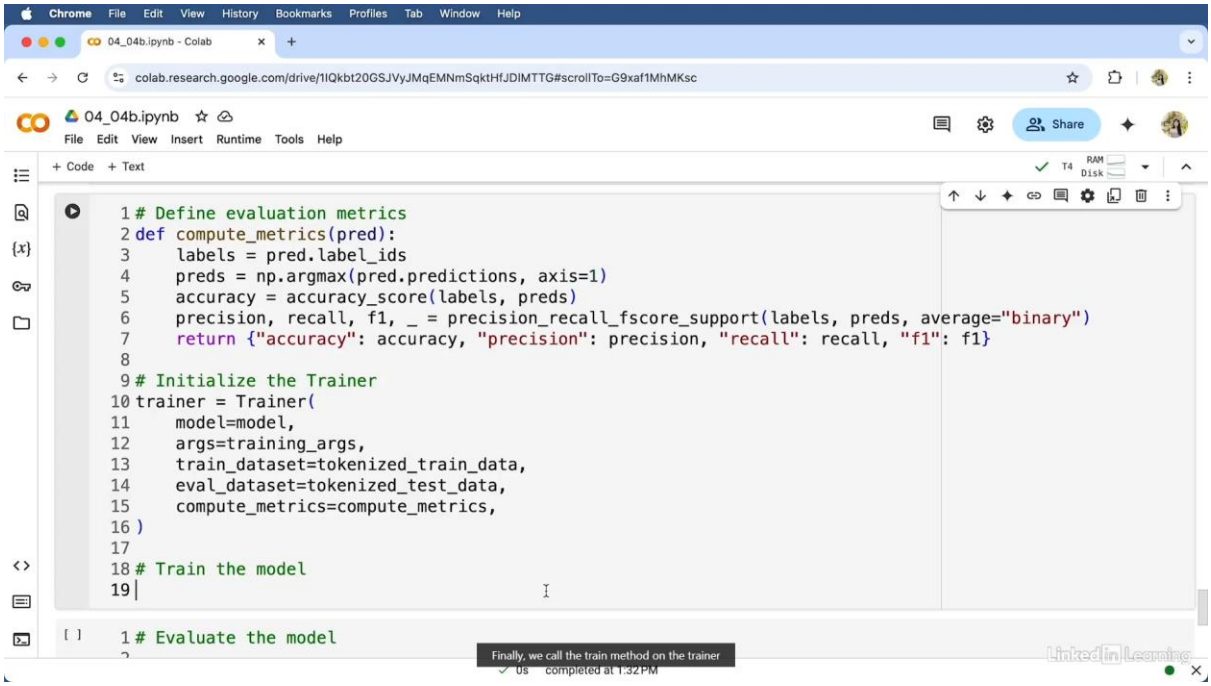
Training the model

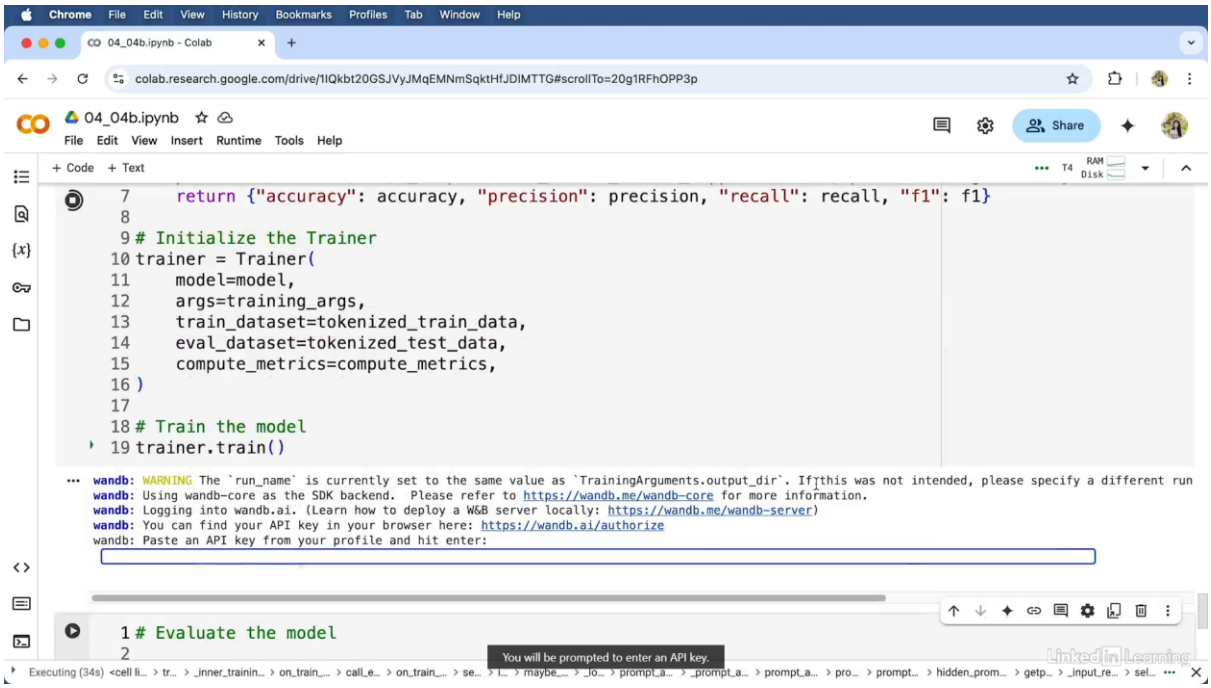
Selecting transcript lines in this section will navigate to timestamp in the video

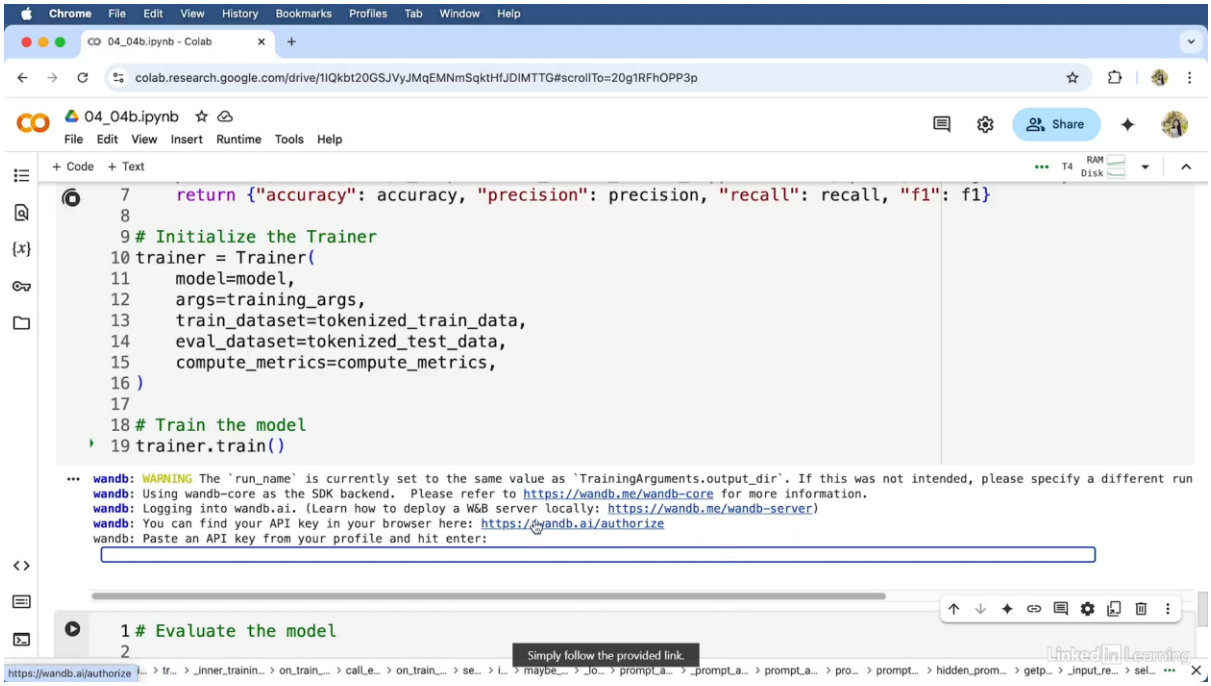
- [Instructor] Before training the model, we define a function called compute_metrics to calculate accuracy, precision, recall, and F1 score. I'll dive deeper into these matrix later, but for now, just know that this function

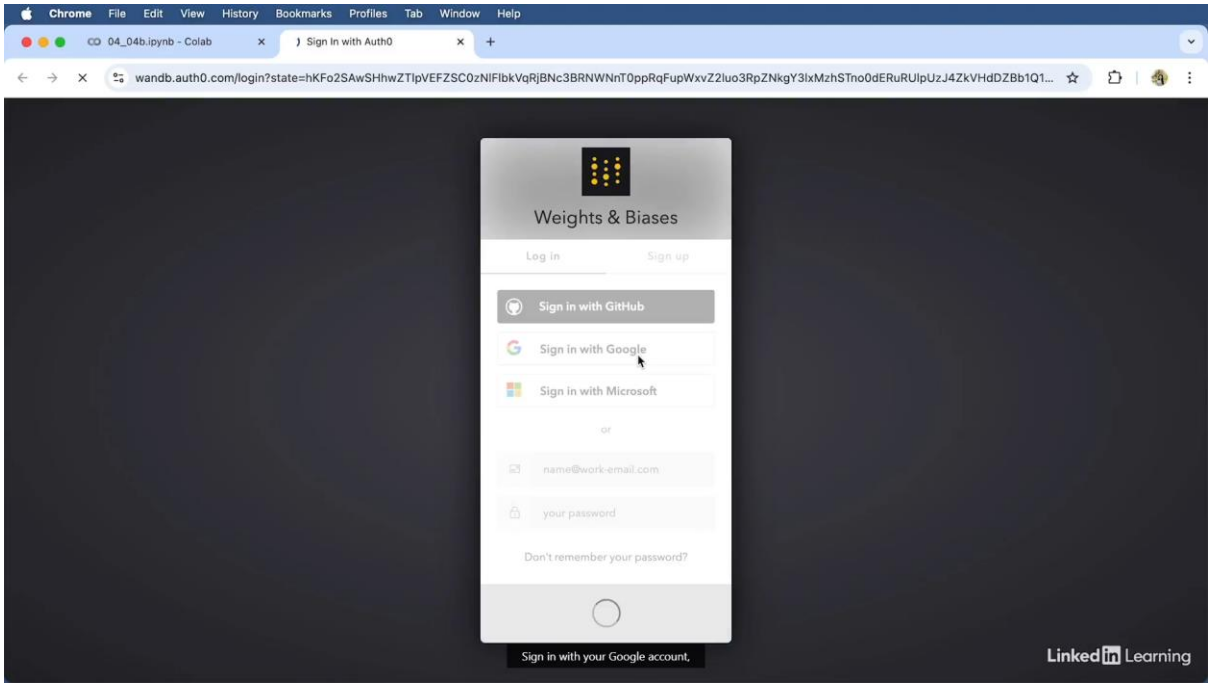
evaluates model performance on both training and test data sets. Next, we initialize the trainer object. It simplifies the training and evaluation process. We pass in the model, training arguments, data sets, and the compute_metrics function. (keyboard clicking) Finally, we call the train method on the trainer to start model training. You will be prompted to enter an API key. Simply follow the provided link. Sign in with your Google account, copy the key, (keyboard clicking) and paste it here. As training proceeds, the model computes and logs evaluation metrics at each step. You'll also see the estimated time remaining, so be patient. Once training is complete, we use trainer.evaluate to print a summary of the evaluation results. We'll analyze those results in the next chapter.

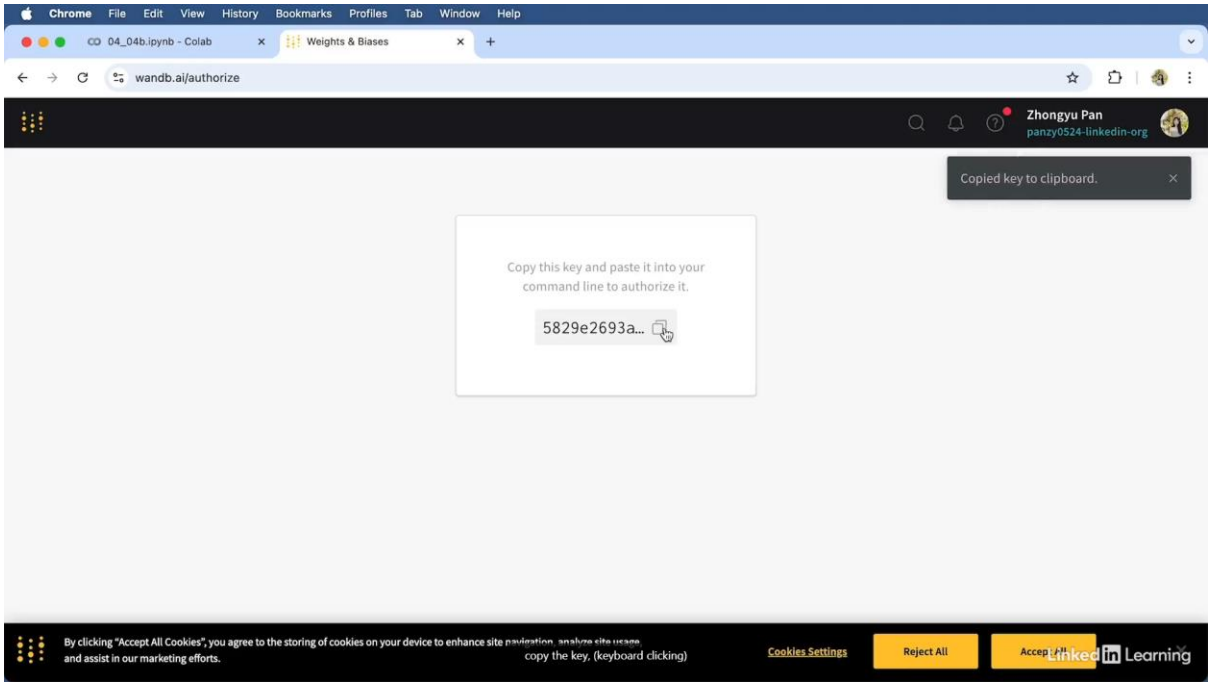


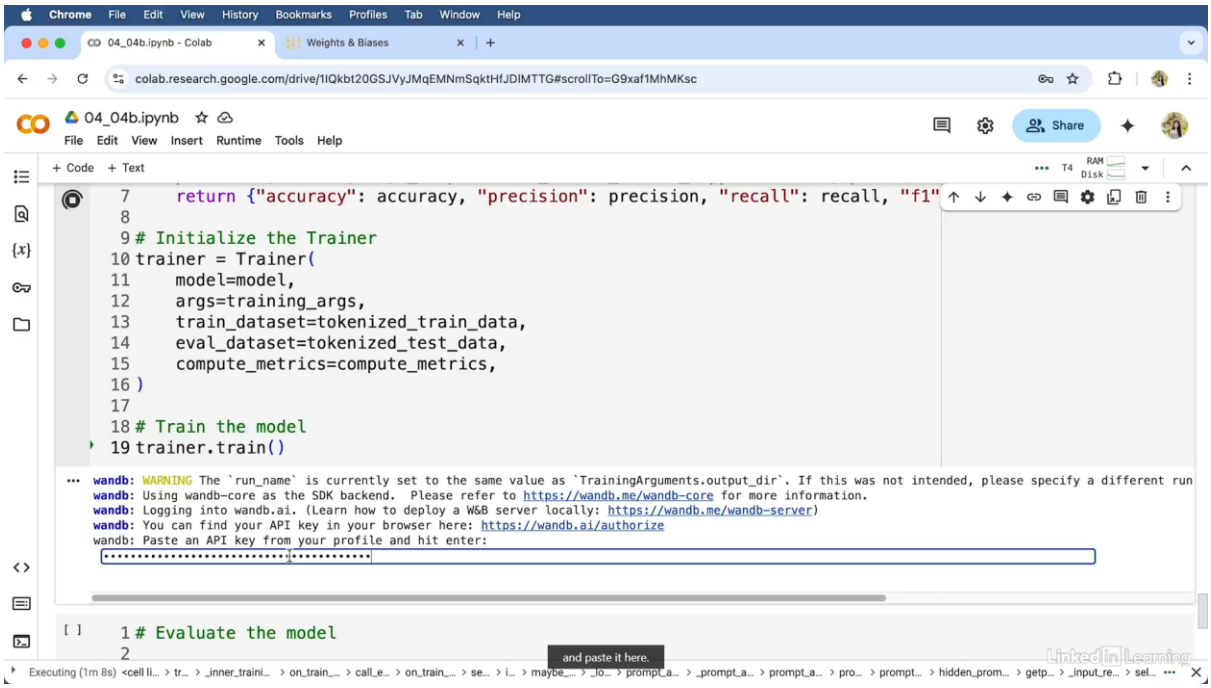

04_04b.ipynb - Colab
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=G9xaf1MhMKsc
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
# Define evaluation metrics
def compute_metrics(pred):
labels = pred.label_ids
preds = np.argmax(pred.predictions, axis=1)
accuracy = accuracy_score(labels, preds)
precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="binary")
return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
# Initialize the Trainer
trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_train_data,
eval_dataset=tokenized_test_data,
compute_metrics=compute_metrics,
)
Train the model
Evaluate the model
Finally, we call the train method on the trainer
0s completed at 1:32PM

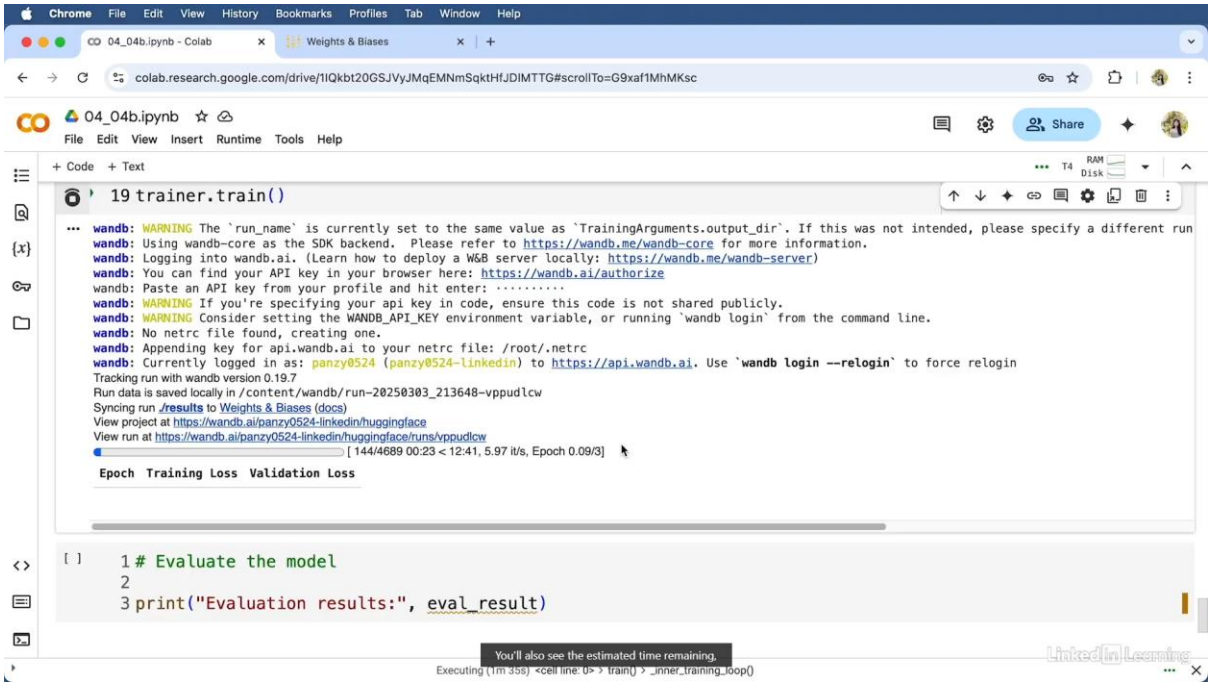

04_04b.ipynb - Colab
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=20g1RFhOPP3p
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
# Initialize the Trainer
trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_train_data,
eval_dataset=tokenized_test_data,
compute_metrics=compute_metrics,
)
# Train the model
trainer.train()
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
Evaluate the model
You will be prompted to enter an API key.

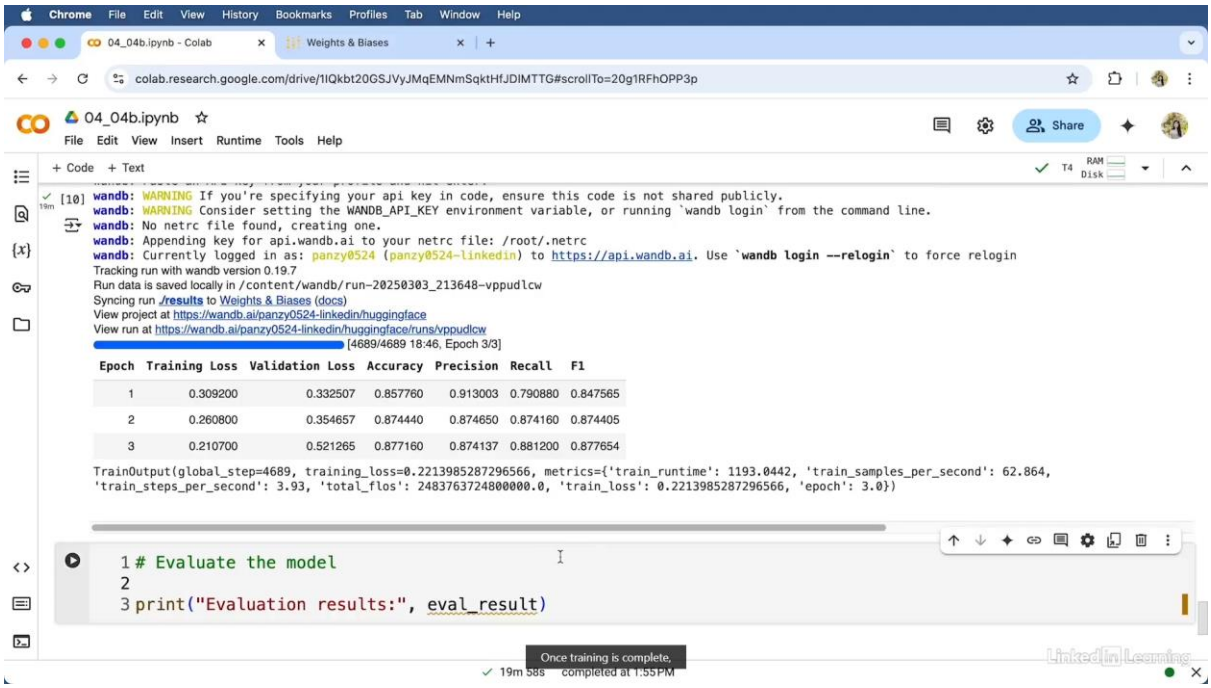

Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDlMTTG#scrollTo=20g1RFhOPP3p
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
# Initialize the Trainer
trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_train_data,
eval_dataset=tokenized_test_data,
compute_metrics=compute_metrics,
)
# Train the model
trainer.train()
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
Evaluate the model
Simply follow the provided link.
https://wandb.ai/authorize
LinkedIn Learning

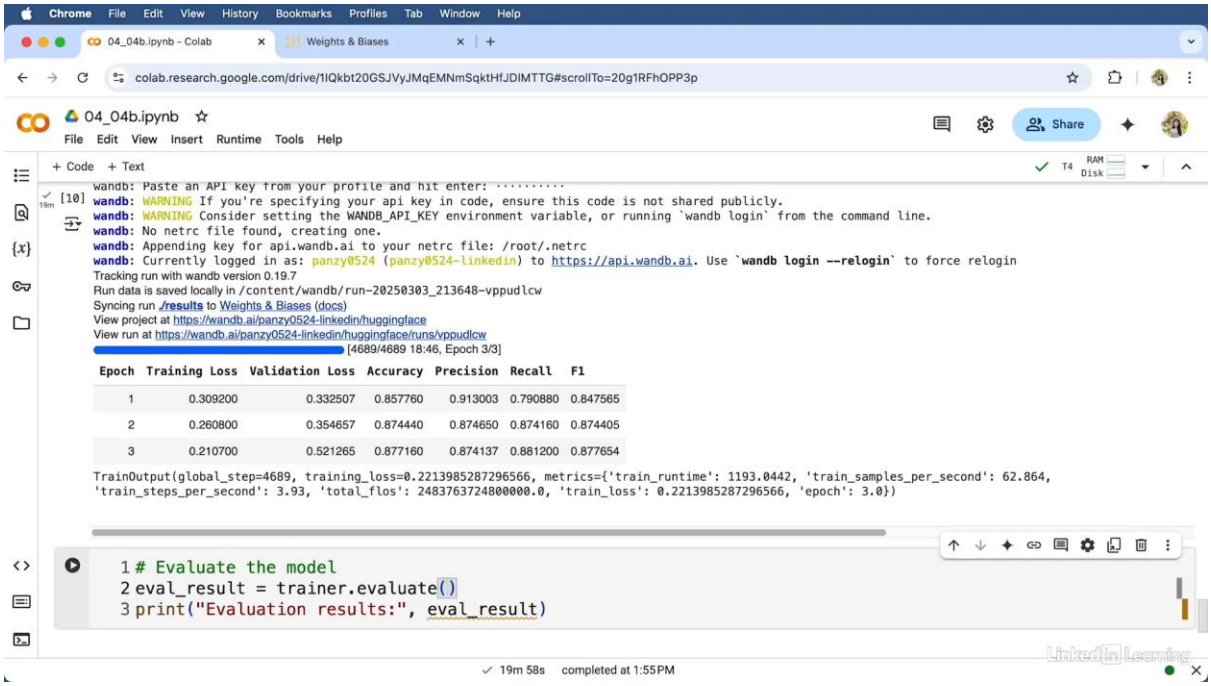

Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Sign In with Auth0
wandb.auth0.com/login?state=hKFo2SAwSHhwZTlpVEFZSC0zNlFlbkVqRjBNc3BRNWNnT0ppRqFupWxvZ2luo3RpZNkgY3lxMzhSTno0dERuRUlpUzJ4ZkVHdDZBb1Q1...
Weights & Biases
Log in
Sign up
Sign in with GitHub
Sign in with Google
Sign in with Microsoft
or
name@work-email.com
your password
Don't remember your password?
Sign in with your Google account.
LinkedIn Learning

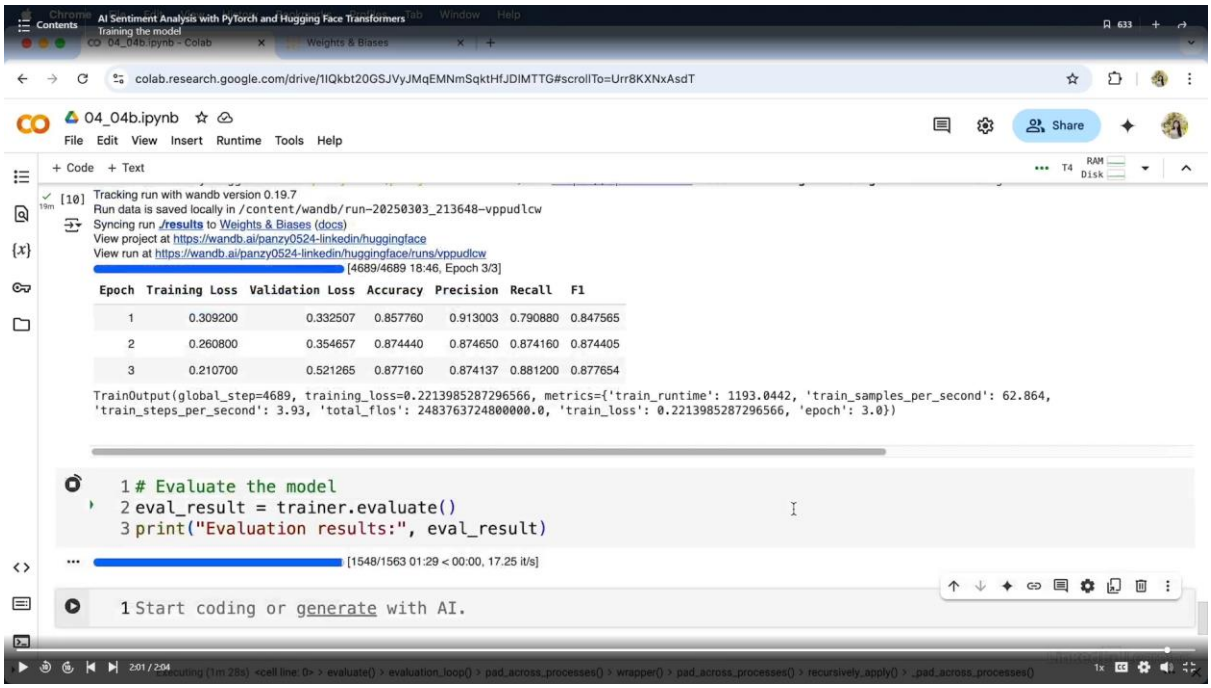

Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Weights & Biases
wandb.ai/authorize
Zhongyu Pan
panzy0524-linkedin-org
Copied key to clipboard.
Copy this key and paste it into your command line to authorize it.
5829e2693a...
By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.
copy the key, (keyboard clicking)
Cookies Settings
Reject All
Accept All
Linked in Learning

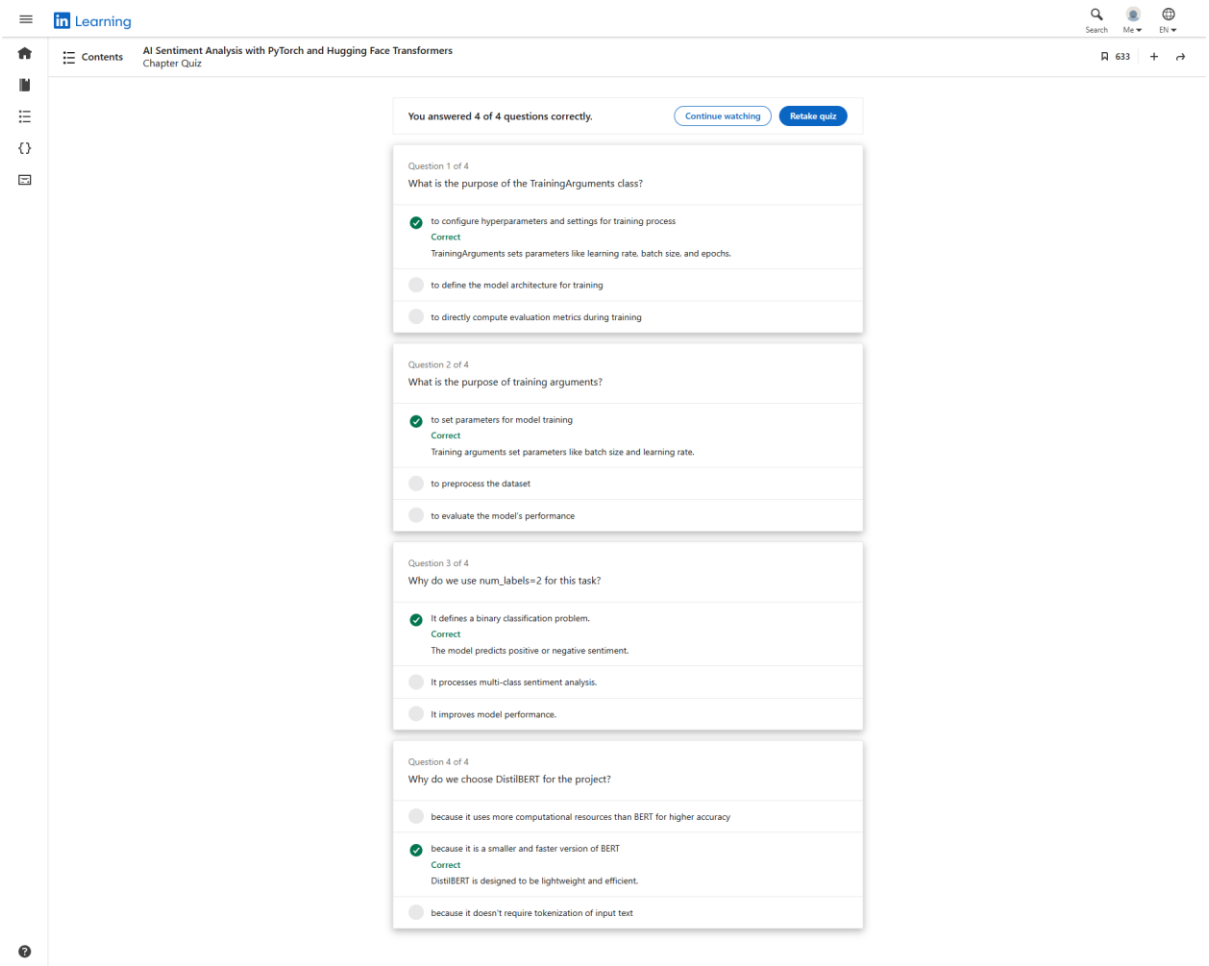

Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Weights & Biases
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=G9xaf1MhMKsc
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
T4
RAM
Disk
return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1"
# Initialize the Trainer
trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_train_data,
eval_dataset=tokenized_test_data,
compute_metrics=compute_metrics,
)
# Train the model
trainer.train()
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
and paste it here.
Evaluate the model
Executing (1m 8s)
Linked in Learning


Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Weights & Biases
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=G9xaf1MhMKsc
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
T4
RAM
Disk
19 trainer.train()
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ·····
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy0524 (panzy0524-linkedin) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw
[144/4689 00:23 < 12:41, 5.97 it/s, Epoch 0.09/3]
Epoch Training Loss Validation Loss
[ ]
1 # Evaluate the model
2
3 print("Evaluation results:", eval_result)
You'll also see the estimated time remaining.
Executing (1m 35s) <cell line: 0> > train() > _inner_training_loop()


Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Weights & Biases
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=20g1RFhOPP3p
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
T4
RAM
Disk
[10]
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy0524 (panzy0524-linkedin) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw
[4689/4689 18:46, Epoch 3/3]
Epoch Training Loss Validation Loss Accuracy Precision Recall F1
1 0.309200 0.332507 0.857760 0.913003 0.790880 0.847565
2 0.260800 0.354657 0.874440 0.874650 0.874160 0.874405
3 0.210700 0.521265 0.877160 0.874137 0.881200 0.877654
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864,
'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
1 # Evaluate the model
2
3 print("Evaluation results:", eval_result)
Once training is complete.
19m 58s completed at 1:55PM


Chrome File Edit View History Bookmarks Profiles Tab Window Help
04_04b.ipynb - Colab
Weights & Biases
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=20g1RFhOPP3p
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
[10]
wandb: Paste an API key from your profile and hit enter: ·····
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy0524 (panzy0524-linkedin) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw
[4689/4689 18:46, Epoch 3/3]
Epoch Training Loss Validation Loss Accuracy Precision Recall F1
1 0.309200 0.332507 0.857760 0.913003 0.790880 0.847565
2 0.260800 0.354657 0.874440 0.874650 0.874160 0.874405
3 0.210700 0.521265 0.877160 0.874137 0.881200 0.877654
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864,
'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
1 # Evaluate the model
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)
19m 58s completed at 1:55PM


Contents
AI Sentiment Analysis with PyTorch and Hugging Face Transformers
Training the model
colab.research.google.com/drive/1IQkbt20GSJVyJMqEMNmSqktHfJDIMTTG#scrollTo=Urr8KXNxAsdT
04_04b.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
[10]
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw
[4689/4689 18:46, Epoch 3/3]
Epoch Training Loss Validation Loss Accuracy Precision Recall F1
1 0.309200 0.332507 0.857760 0.913003 0.790880 0.847565
2 0.260800 0.354657 0.874440 0.874650 0.874160 0.874405
3 0.210700 0.521265 0.877160 0.874137 0.881200 0.877654
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864,
'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
1 # Evaluate the model
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)
[1548/1563 01:29 < 00:00, 17.25 it/s]
1 Start coding or generate with AI.
2:01 / 2:04

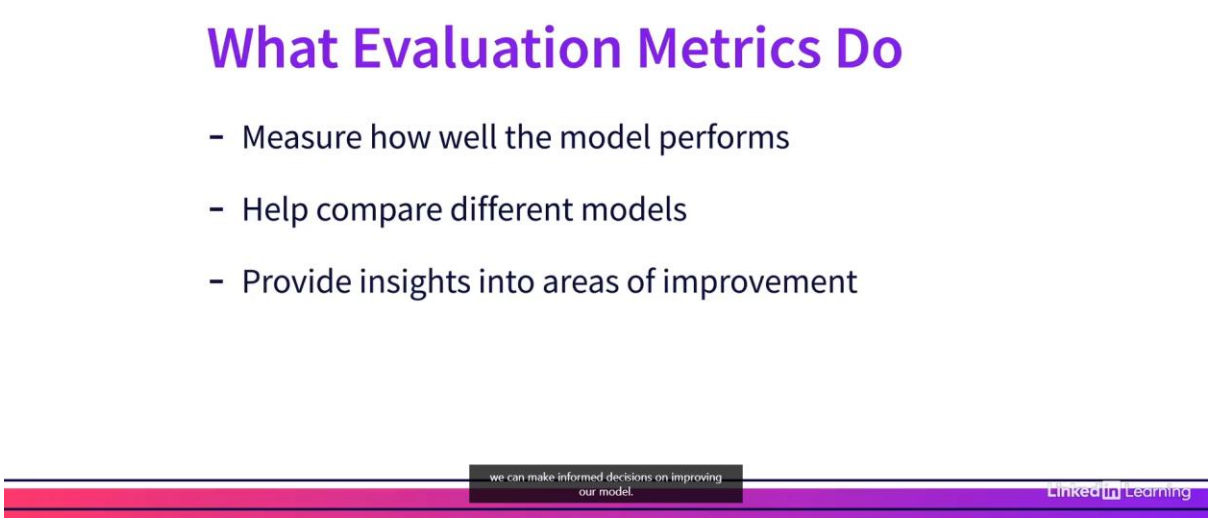


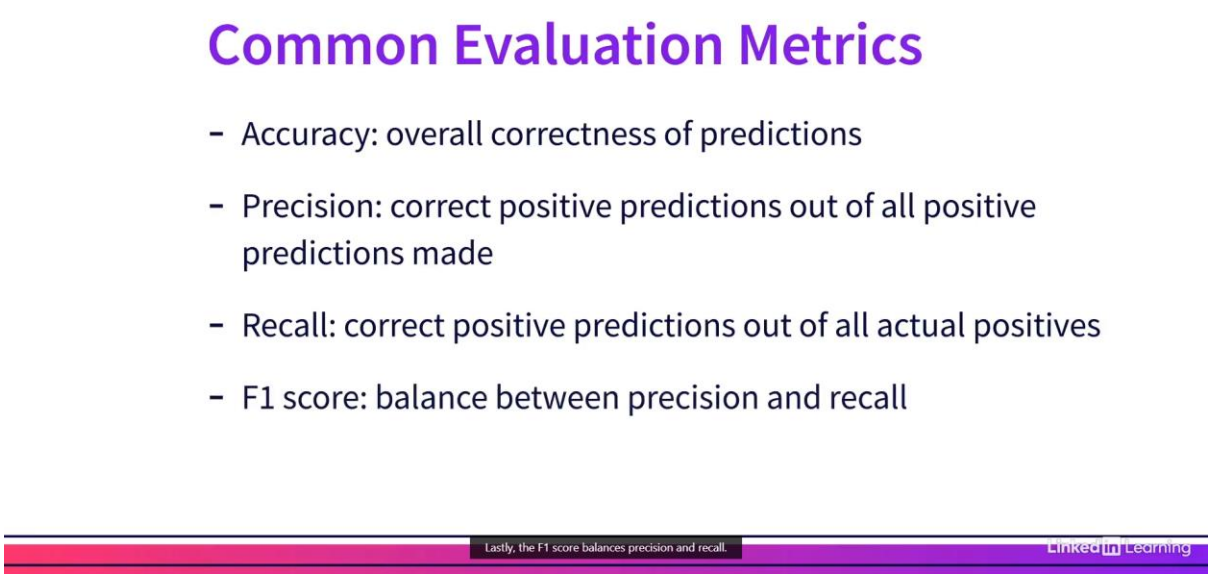
Introduction to evaluation metrics

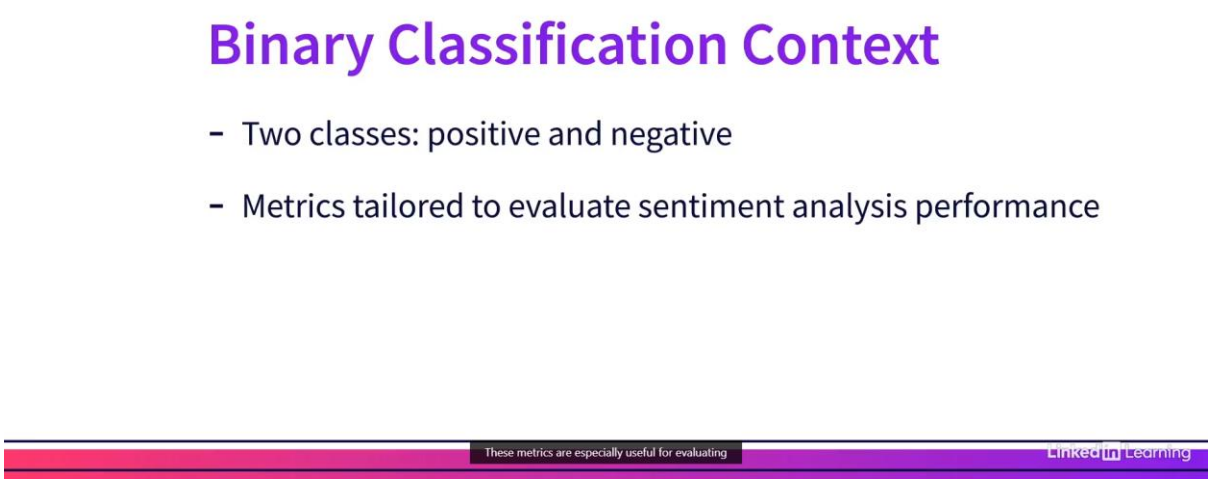
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this chapter, we'll look at some commonly used evaluation metrics. Evaluation metrics tell us how well a model performs. This allow us to compare different models. By analyzing these metrics, we can make informed decisions on improving our model. Let's go over four key metrics. First is accuracy. It represents the percentage of correctly predicted labels. Second is precision. It measures how many of the predicted positives are actually correct. Recall focuses on the percentage of actual positives that were correctly identified. Lastly, the F1 score balances precision and recall. It provides a more comprehensive measure of model performance. In sentiment analysis, our task is a binary classification. Identifying positive or

negative sentiment. These metrics are especially useful for evaluating how well our model distinguishes between the two classes.





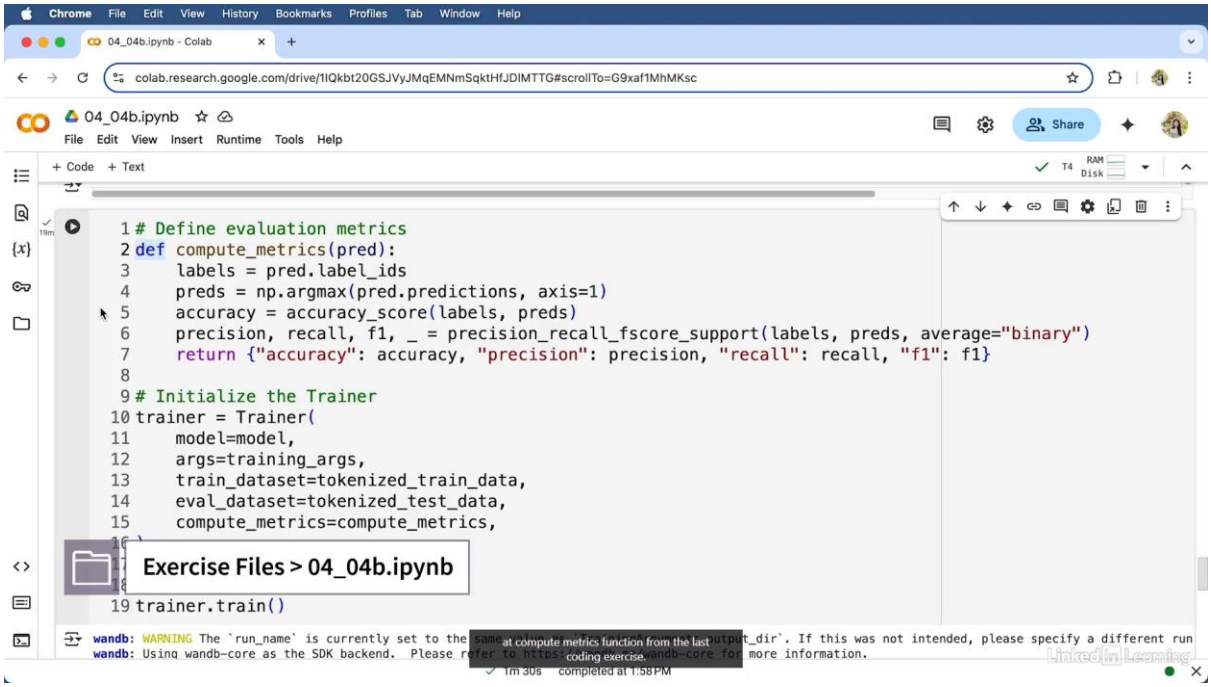


Calculating accuracy, precision, recall, and F1 score

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll take a closer look at compute metrics function from the last coding exercise. There won't be any new coding session in this video. The compute metrics function calculates four key evaluation metrics. Let's break it down step by step-by-step. First, we extract

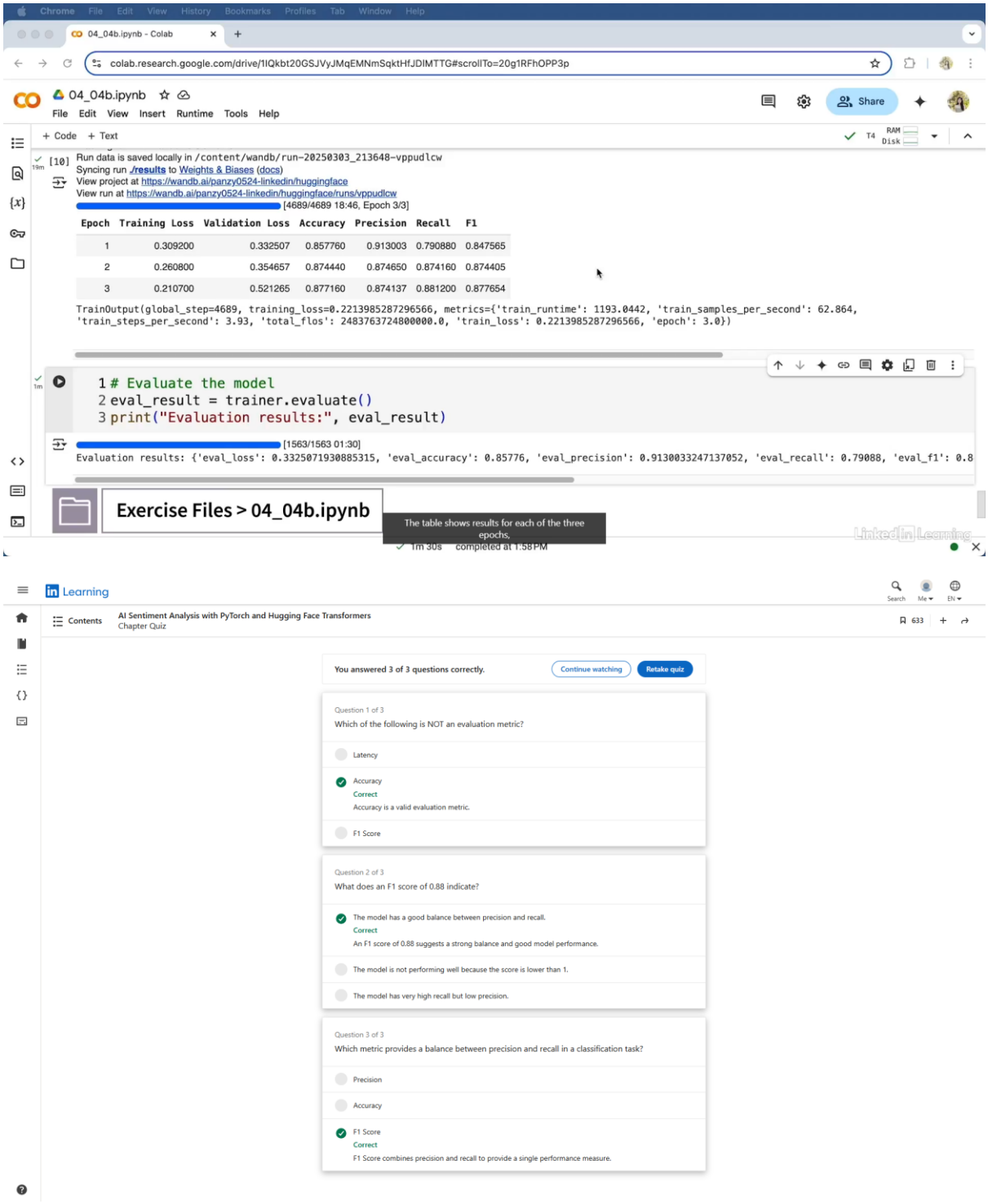
label IDs from the predictions object into the variable called labels. These are the true sentiment labels from our dataset, which we'll compare against the model's predictions. Next, we calculate preds or model predictions by applying the argmax function, which selects the class with the highest prediction scores. Now, the function calculates the four key metrics we introduced in the last video. I encourage you to implement them yourself based on their definitions. Finally, the function returns these metrics as a dictionary.



Analyzing results

Selecting transcript lines in this section will navigate to timestamp in the video

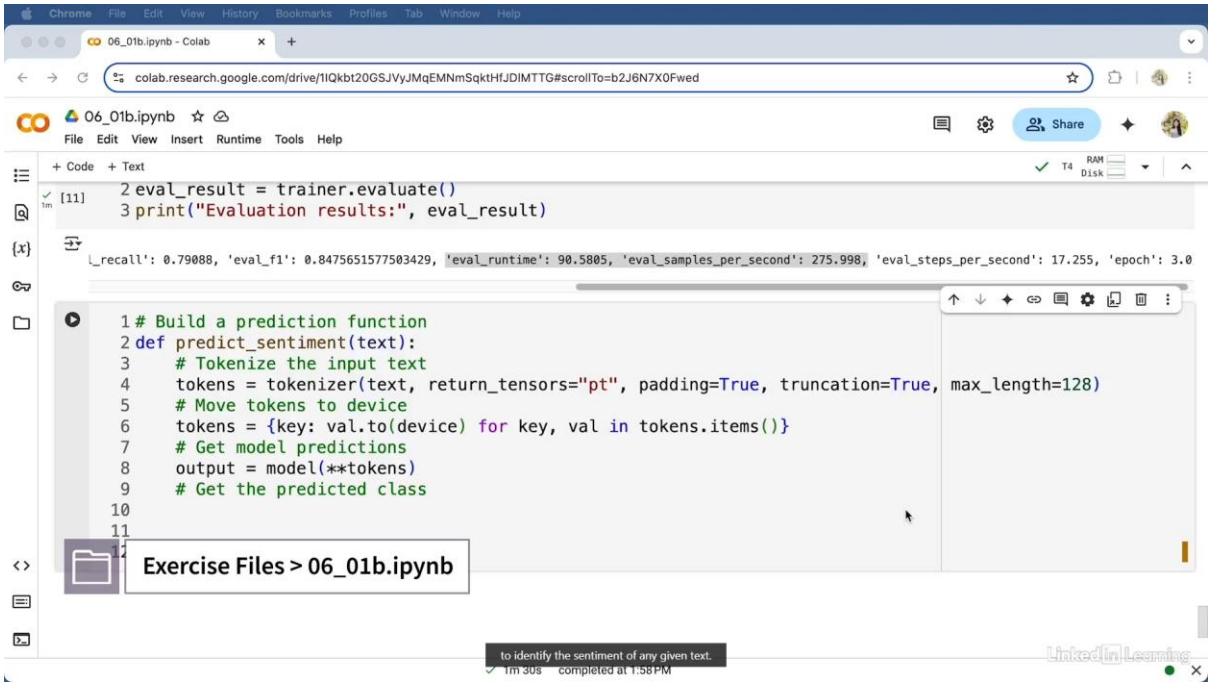
- [Instructor] In this video, we'll look at evaluation results. The table shows results for each of the three epochs, but we'll only focus on the overall evaluation results here. The overall results show the model performs well with an accuracy of 85.776%, along with relatively high precision recall and F1 score. The model also processes data efficiently with a round time of 91 seconds and a throughput of 276 samples per second. These matrix give us confidence in the model's ability to classify movie reviews as positive or negative.

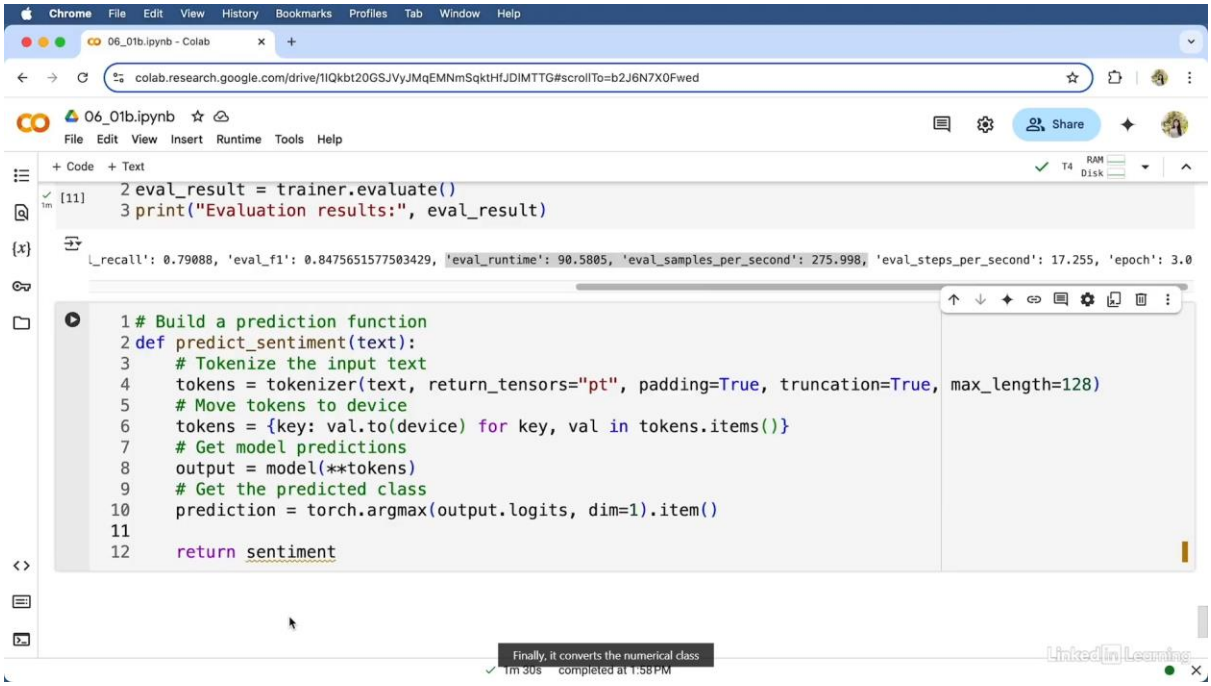


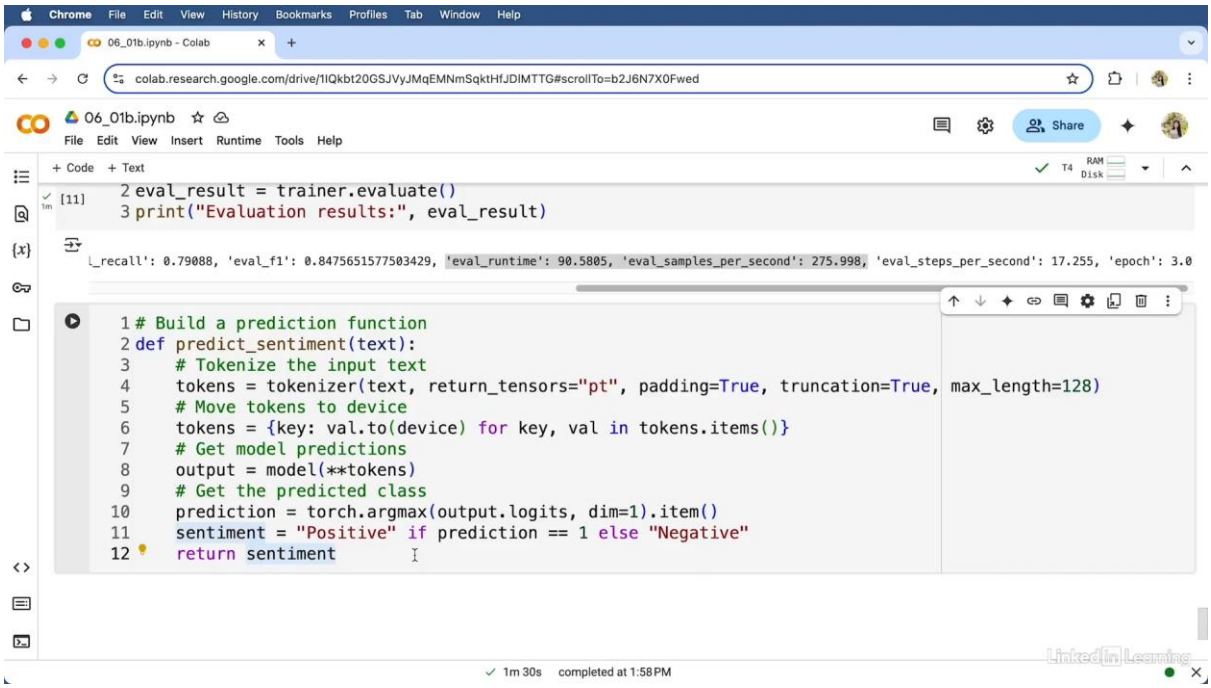
Let's build a prediction function

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this section, we'll create a prediction function to identify the sentiment of any given text. The function predict_sentiment takes an input text, processes it, and returns whether the sentiment is positive or negative. The function tokenizes the input text, applies padding and truncation, and moves the tokens to the same device as the model. The model then processes them and produces raw prediction scores. Next, the function identifies the class with the highest score using torch.argmax. Finally, it converts the numerical class into a sentiment label, positive for class 1 and a negative for class 0. Then return the sentiment.





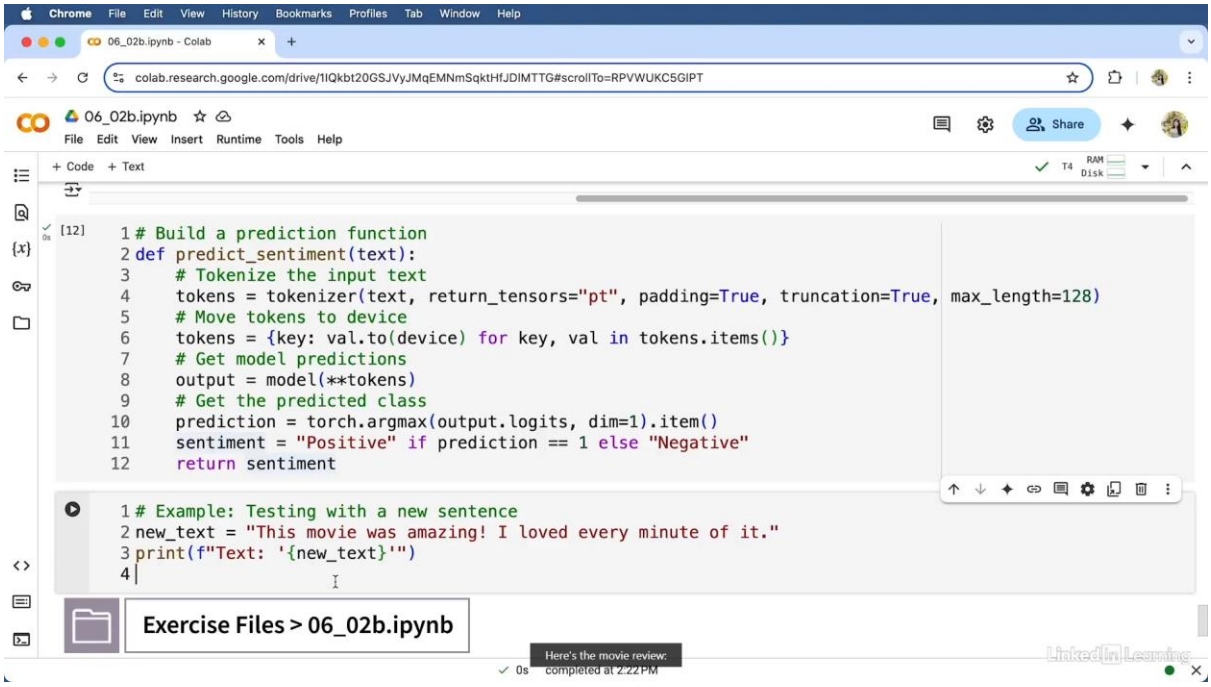


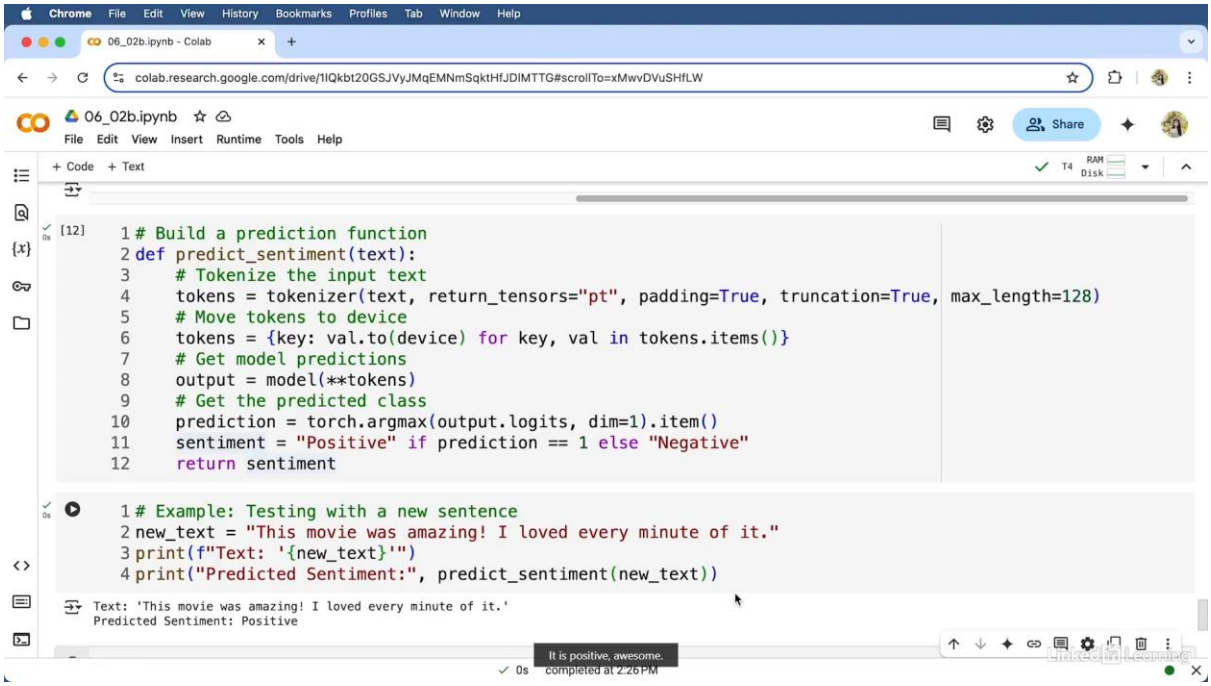
Test the model yourself

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's test our model with a real example using print sentiment function. Here's the movie review: "This movie was amazing! I loved every minute of it." Clearly we expect the output label to be positive. We pass this

review to predict sentiment function, and the model will tell us if it's positive or negative. Let's print out the predicted sentiment. It is positive, awesome. Here comes the exciting part. Try testing it yourself with different examples and see how well the model captures the tone of movie reviews.

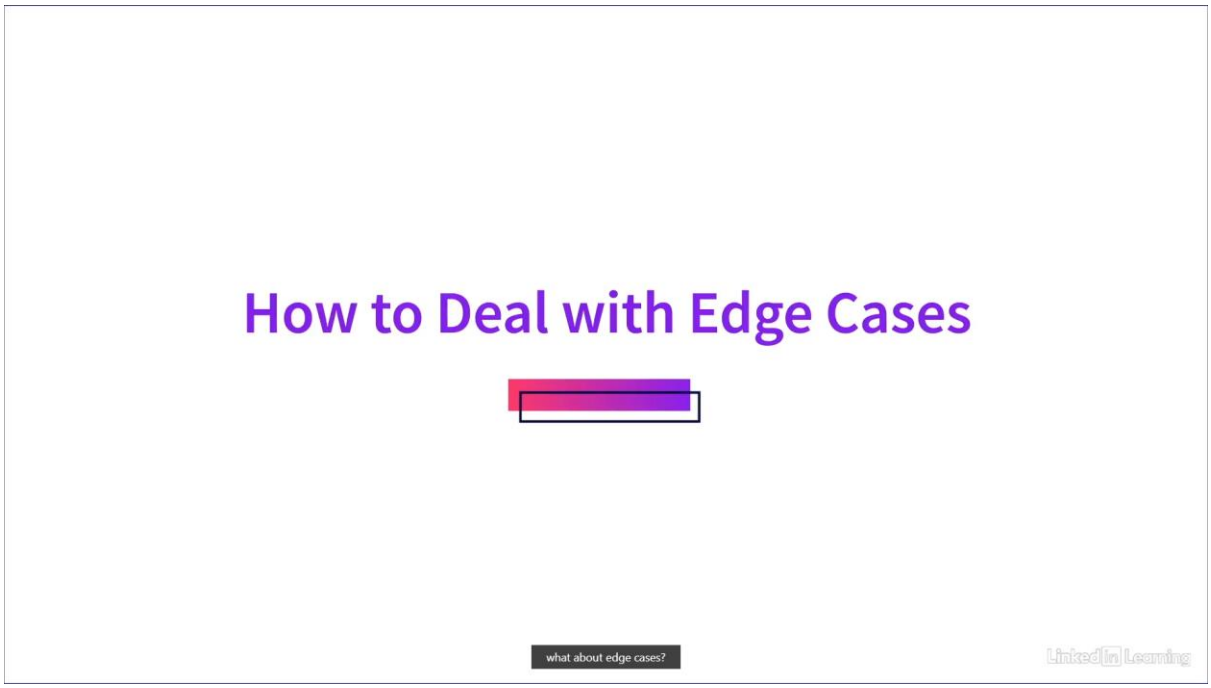




How to deal with edge cases

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] You might be wondering, what about edge cases? For example, consider sentences with mixed sentiments, like, "The movie was visually stunning, but the story was boring." These inputs contain both positive and negative elements, which can confuse the model. Another challenge is sarcasm, such as, "Oh, great, another masterpiece of boredom." Detecting sarcasm requires a deeper understanding of context. Handling edge cases often requires fine-tuning on specific datasets, preprocessing tricky inputs, or combining sentiment analysis with techniques like contextual embeddings or sentiment lexicons. I encourage you to explore these topics further to see how they help manage edge cases. When building applications, it's important to test thoroughly on a diverse dataset to identify potential weaknesses. And remember, no model is perfect, but understanding its limitations helps us improve.



Edge Cases

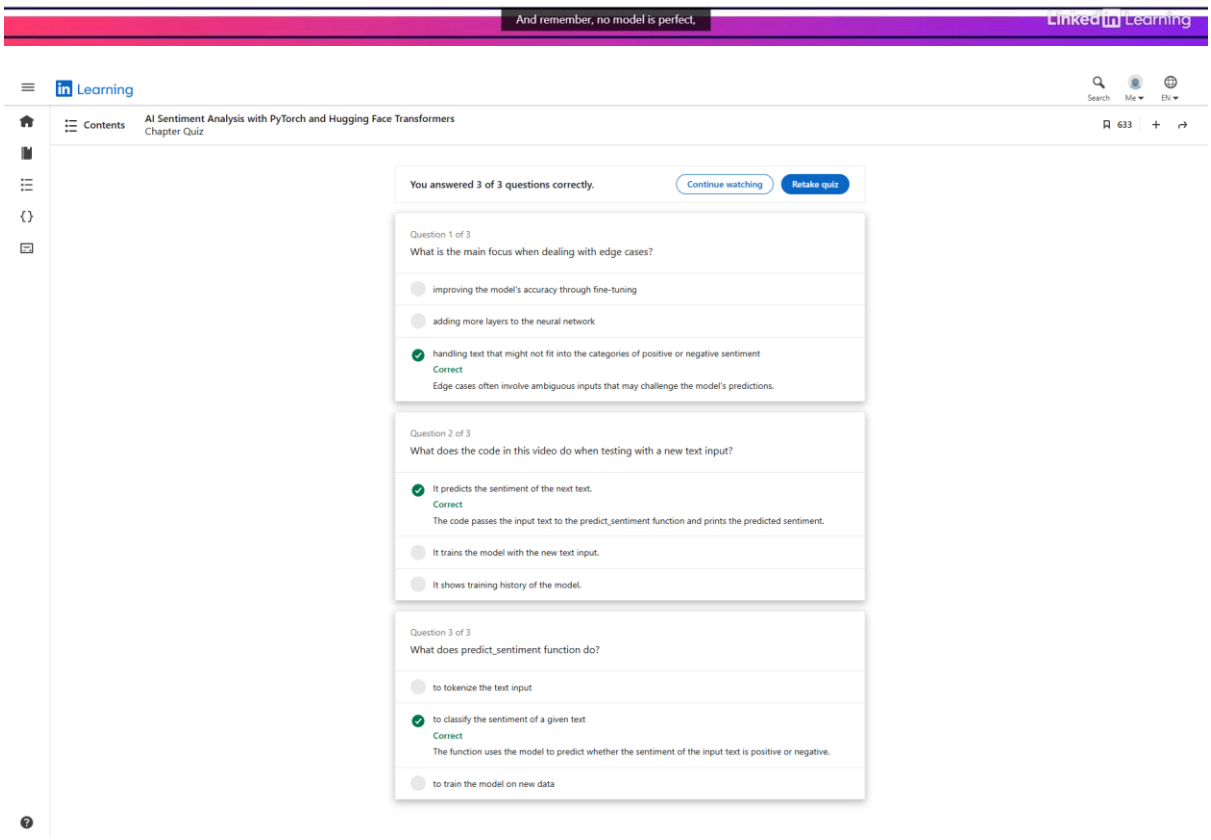
- Mixed sentiments
- Sarcasm

"Oh, great, another masterpiece of boredom."

How to Deal with Edge Cases

- Fine-tuning the model on specific data
- Preprocessing to handle tricky inputs
- Combining sentiment analysis with other techniques

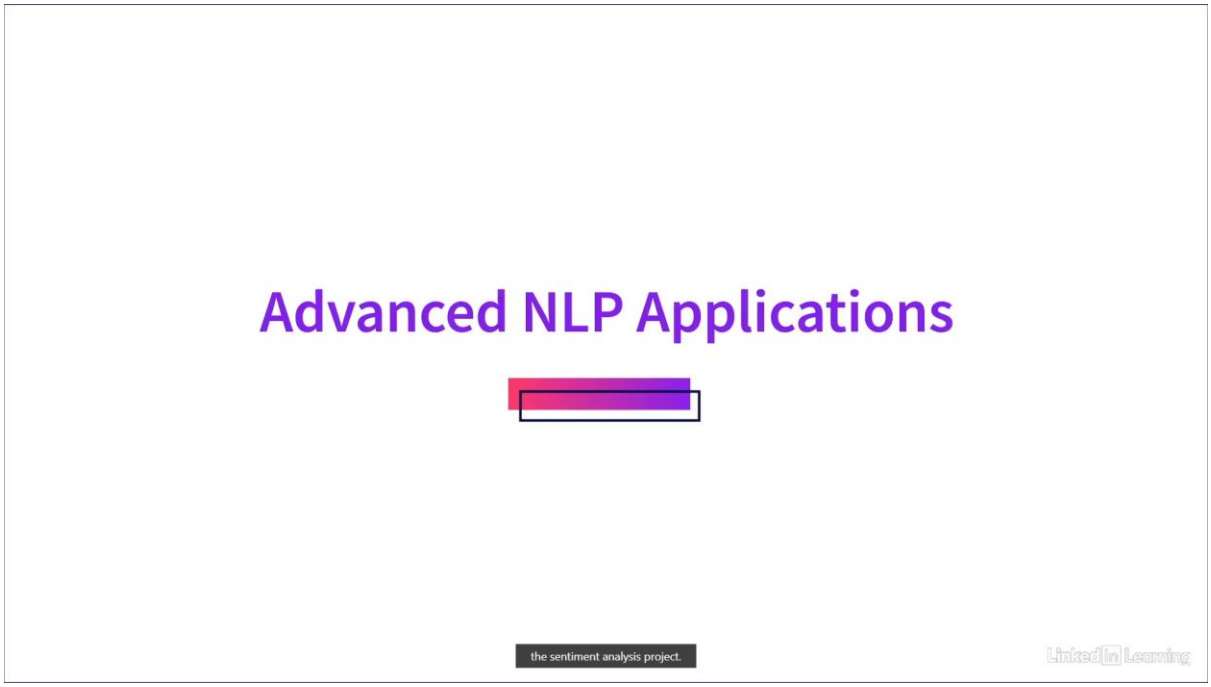
or combining sentiment analysis with techniques



Advanced NLP applications

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Great job on completing the sentiment analysis project. Now let's take a break and check out some other exciting applications in NLP. Text summarization is a game changer for handling large amounts of information. Imagine reading a paper and using NLP to generate a short, concise summary. This is especially useful for news, research and any content where you need the key points without going through everything. Next is machine translation. Tools like Google Translate use NLP models to break down language barriers. These models enable real time translation, making it easier to communicate with people across the globe. Have you ever interacted with a chat bot? It is powered by question answering models. These models process questions and generate answers from customer support to retrieving information online. QA systems are great for delivering helpful and accurate responses.



Text Summarization

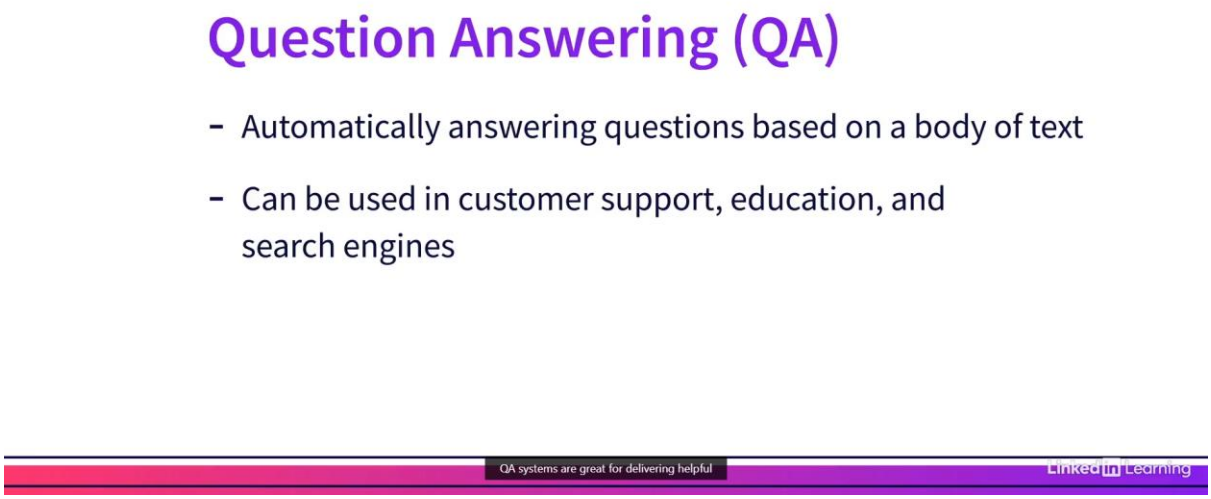
- Automatically generating concise summaries of long documents
- Useful for news, research papers, and content curation

This is especially useful for news, research

Machine Translation

- Translating text from one language to another using AI models
- Examples include Google Translate and other multilingual applications

These models enable real time translation, making it easier



Continue your AI sentiment analysis journey

Selecting transcript lines in this section will navigate to timestamp in the video

- Congratulations on completing the course. You now have a solid understanding and hands-on practice with sentiment analysis. This is just the beginning. I can't wait to see how you apply the skill you've learned to real-world projects. There's so much more to discover in the world of NLP. Stay curious and keep learning through online resources, research papers and NLP communities. And keep an eye out for more courses from me on advanced NLP applications. I hope you're excited to put your new skills into practice. Keep up the great work, and I look forward to seeing you soon.