**Fine-tune Phi-2 for Sentiment Analysis**

Phi-2 is a Transformer with 2.7 billion parameters. It was trained using the same data sources as Phi-1.5, augmented with a new data source that consists of various NLP synthetic texts and filtered websites (for safety and educational value). When assessed against benchmarks testing common sense, language understanding, and logical reasoning, Phi-2 showcased a nearly state-of-the-art performance among models with less than 13 billion parameters.

This makes Phi-2 one of the best performing small transformer model till date.

**Case Study**

Sentiment analysis on financial and economic information is highly relevant for businesses for several key reasons, ranging from market insights (gain valuable insights into market trends, investor confidence, and consumer behavior) to risk management (identifying potential reputational risks) to investment decisions (gauging the sentiment of stakeholders, investors, and the general public businesses can assess the potential success of various investment opportunities).

Before the technicalities of fine-tuning a large language model like Phi-2, we had to find the correct dataset to demonstrate the potentialities of fine-tuning.

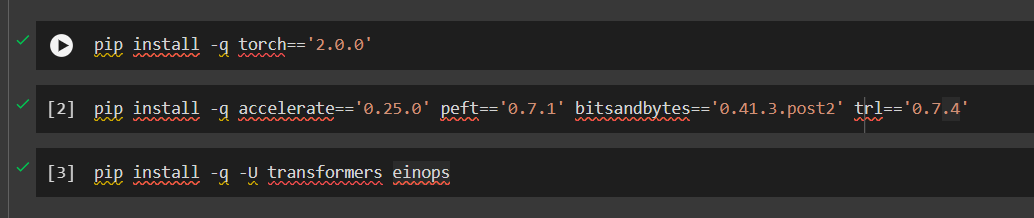
Particularly within the realm of finance and economic texts, annotated datasets are notably rare, with many being exclusively reserved for proprietary purposes. To address the issue of insufficient training data, scholars from the Aalto University School of Business introduced in 2014 a set of approximately 5000 sentences. This collection aimed to establish human-annotated benchmarks, serving as a standard for evaluating alternative modeling techniques. The involved annotators (16 people with adequate background knowledge on financial markets) were instructed to assess the sentences solely from the perspective of an investor, evaluating whether the news potentially holds a positive, negative, or neutral impact on the stock price.

**Data Set**

The FinancialPhraseBank dataset is a comprehensive collection that captures the sentiments of financial news headlines from the viewpoint of a retail investor. Comprising two key columns, namely "Sentiment" and "News Headline," the dataset effectively classifies sentiments as either negative, neutral, or positive. This structured dataset serves as a valuable resource for analyzing and understanding the complex dynamics of sentiment in the domain of financial news. It has been used in various studies and research initiatives, since its inception in the work by Malo, P., Sinha, A., Korhonen, P., Wallenius, J., and Takala, P. "Good debt or bad debt: Detecting semantic orientations in economic texts.", published in the Journal of the Association for Information Science and Technology in 2014.

**Dependencies**

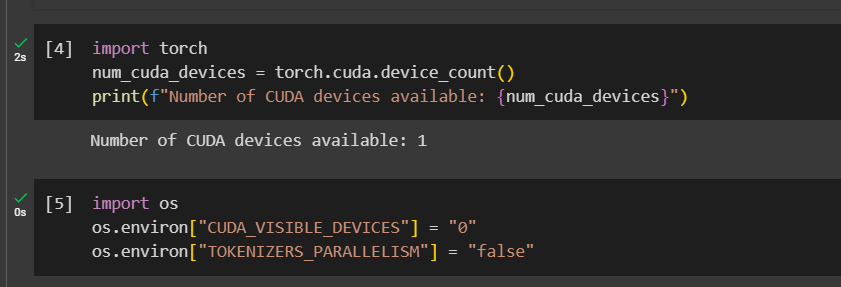
* accelerate is a distributed training library for PyTorch by HuggingFace. It allows you to train your models on multiple GPUs or CPUs in parallel (distributed configurations), which can significantly speed up training in presence of multiple GPUs (we won't use it in our example since we are using colab notebook with a free tier GPU).
* peft is a Python library by HuggingFace for efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs.
* bitsandbytes by Tim Dettmers, is a lightweight wrapper around CUDA custom functions, in particular 8-bit optimizers, matrix multiplication (LLM.int8()), and quantization functions. It allows to run models stored in 4-bit precision: while 4-bit bitsandbytes stores weights in 4-bits, the computation still happens in 16 or 32-bit and here any combination can be chosen (float16, bfloat16, float32, and so on).
* transformers is a Python library for natural language processing (NLP). It provides a number of pre-trained models for NLP tasks such as text classification, question answering, and machine translation.
* trl is a full stack library by HuggingFace providing a set of tools to train transformer language models with Reinforcement Learning, from the Supervised Fine-tuning step (SFT), Reward Modeling step (RM) to the Proximal Policy Optimization (PPO) step.

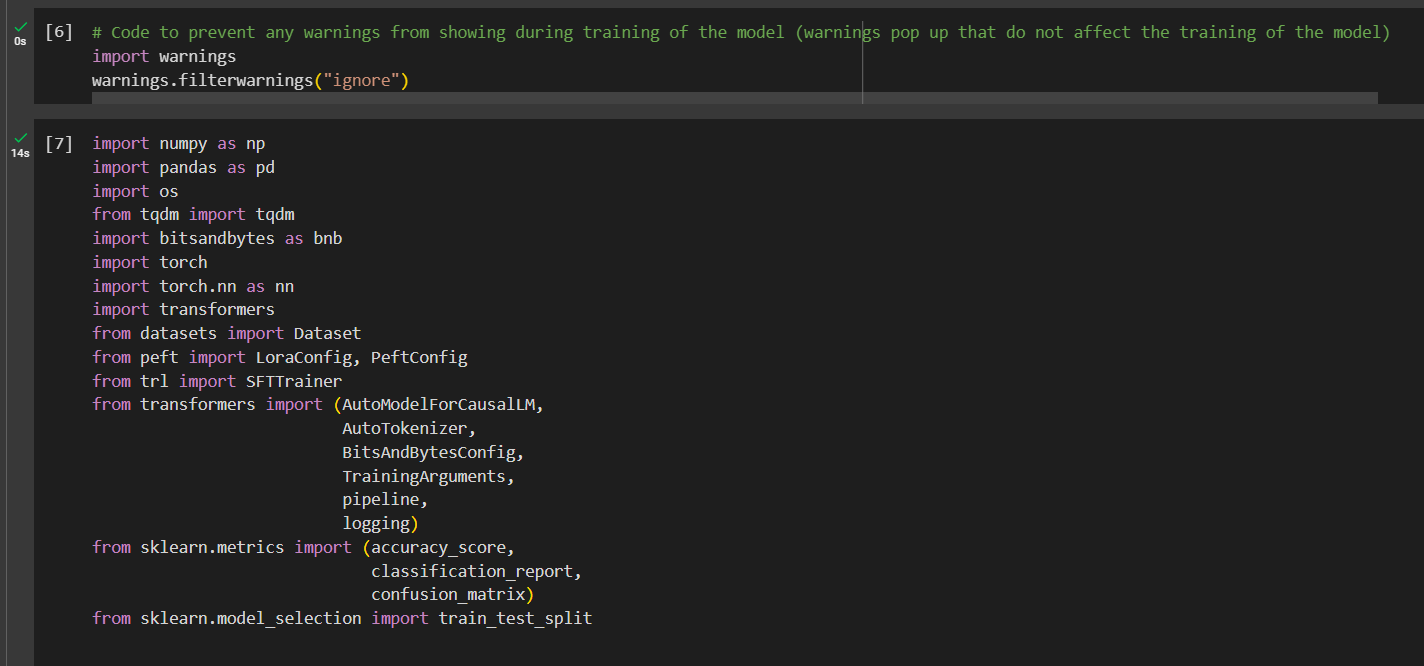


**Code Walkthrough**

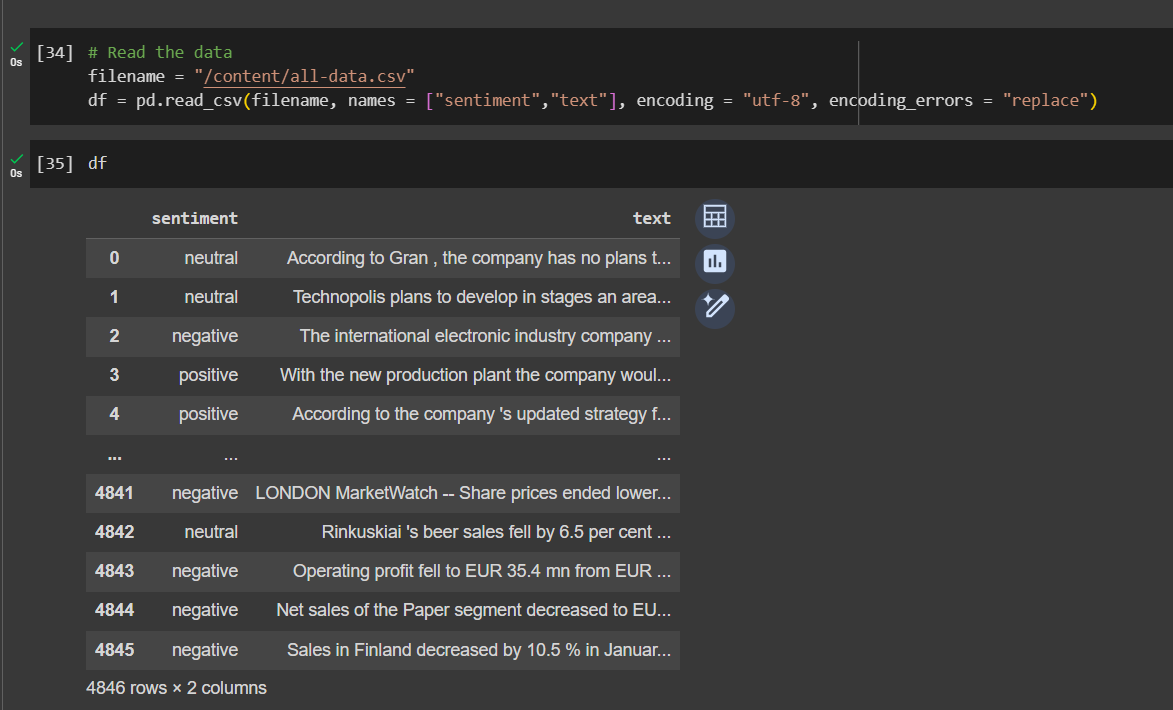
The below code imports the os module and sets two environment variables:

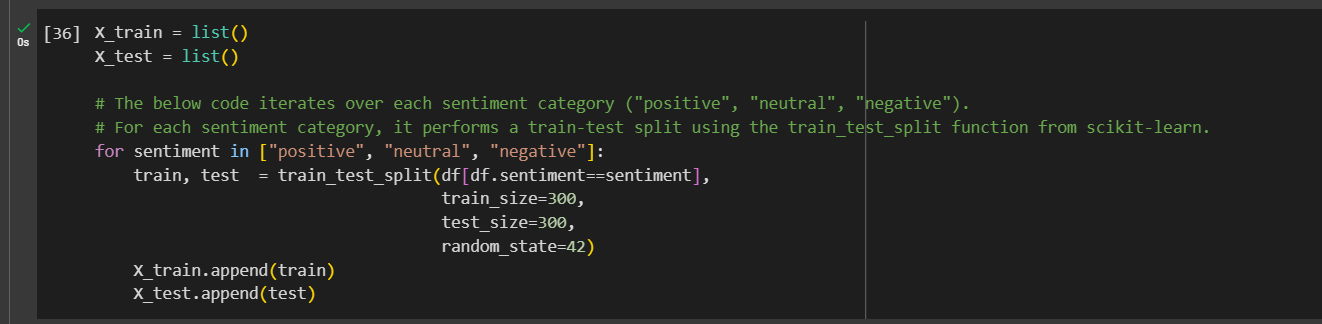
* CUDA\_VISIBLE\_DEVICES: This environment variable tells PyTorch which GPUs to use. In this case, the code is setting the environment variable to 0, which means that PyTorch will use the first GPU.
* TOKENIZERS\_PARALLELISM: This environment variable tells the Hugging Face Transformers library whether to parallelize the tokenization process. In this case, the code is setting the environment variable to false, which means that the tokenization process will not be parallelized.

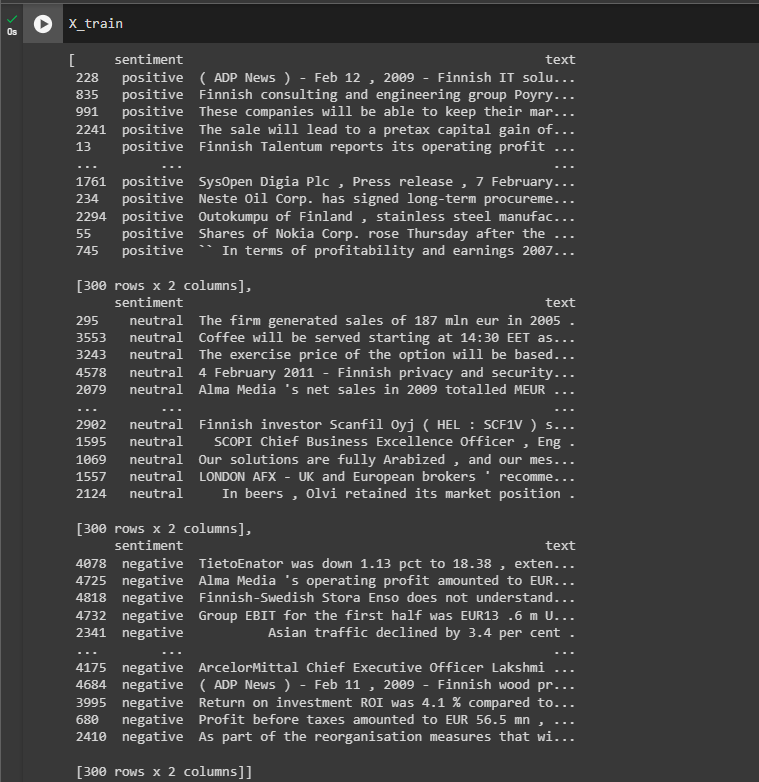




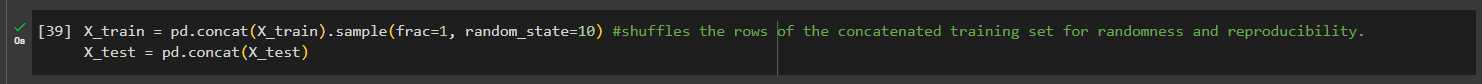
**Dataset Preparation**

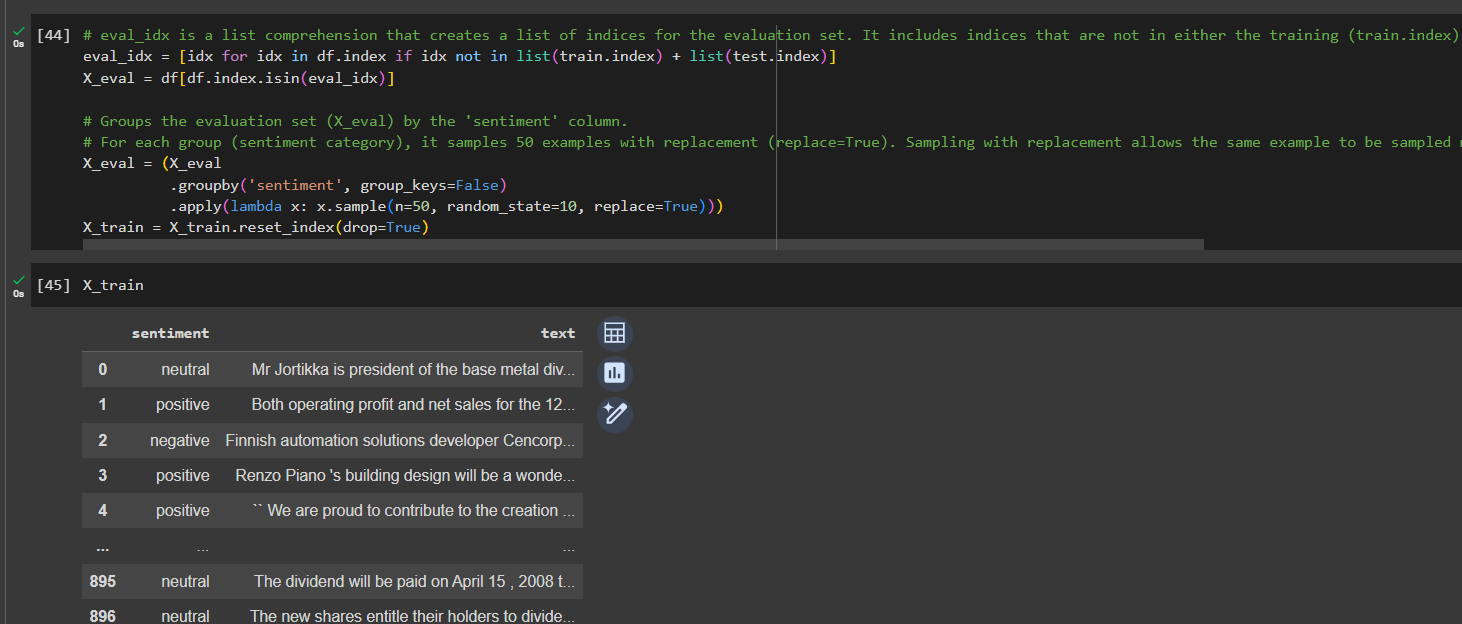






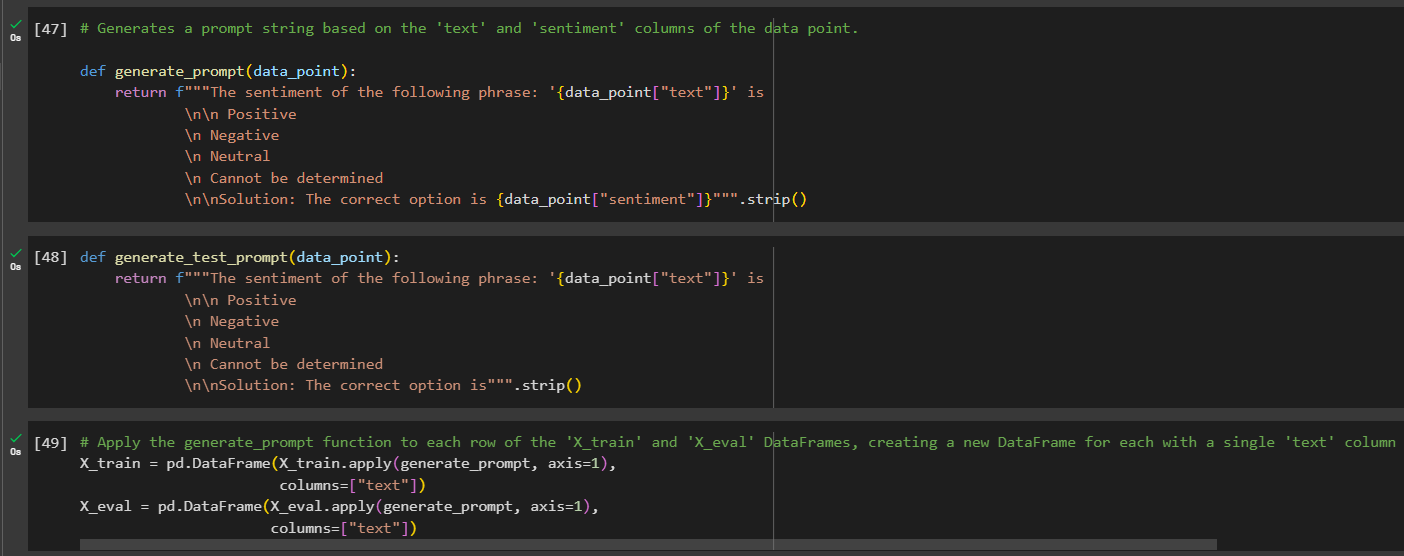


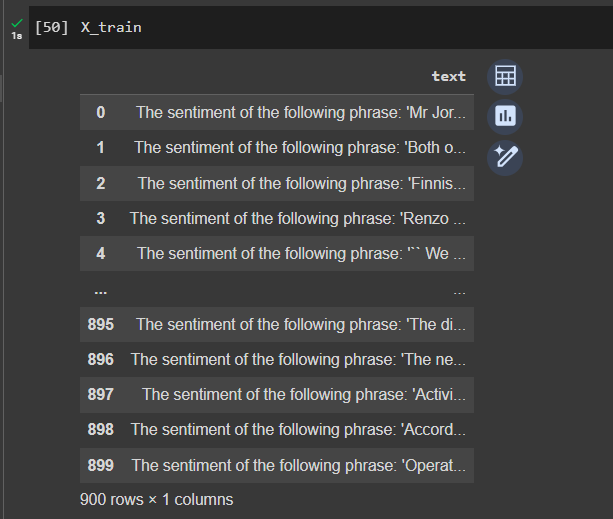


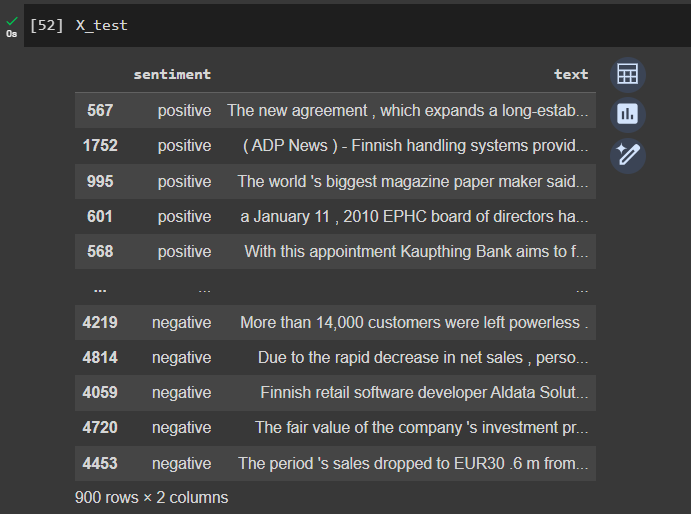


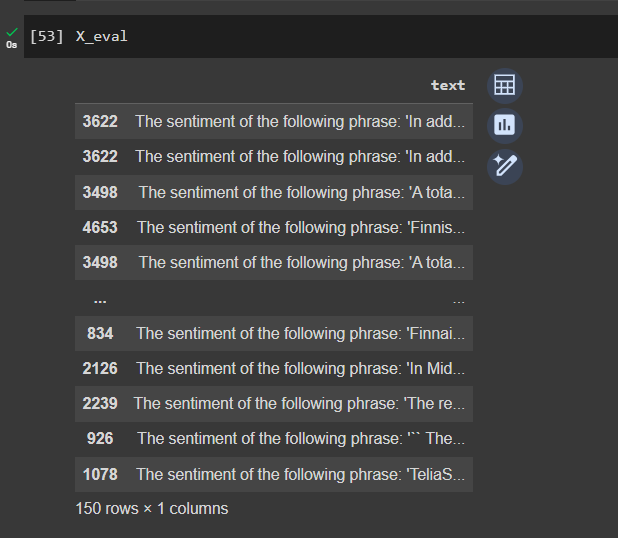
Now we use a prompt string to generate a data set with a single “text” column to train the model on. The train prompts contains the expected answer we want to fine-tune the model with,

We also create an evaluation set. The residual examples not in train or test, for reporting purposes during training (but it won't be used for early stopping), is treated as evaluation data, which is sampled with repetition in order to have a 50/50/50 sample (negative instances are very few, hence they should be repeated). The train and eval data are wrapped by the class from Hugging Face.



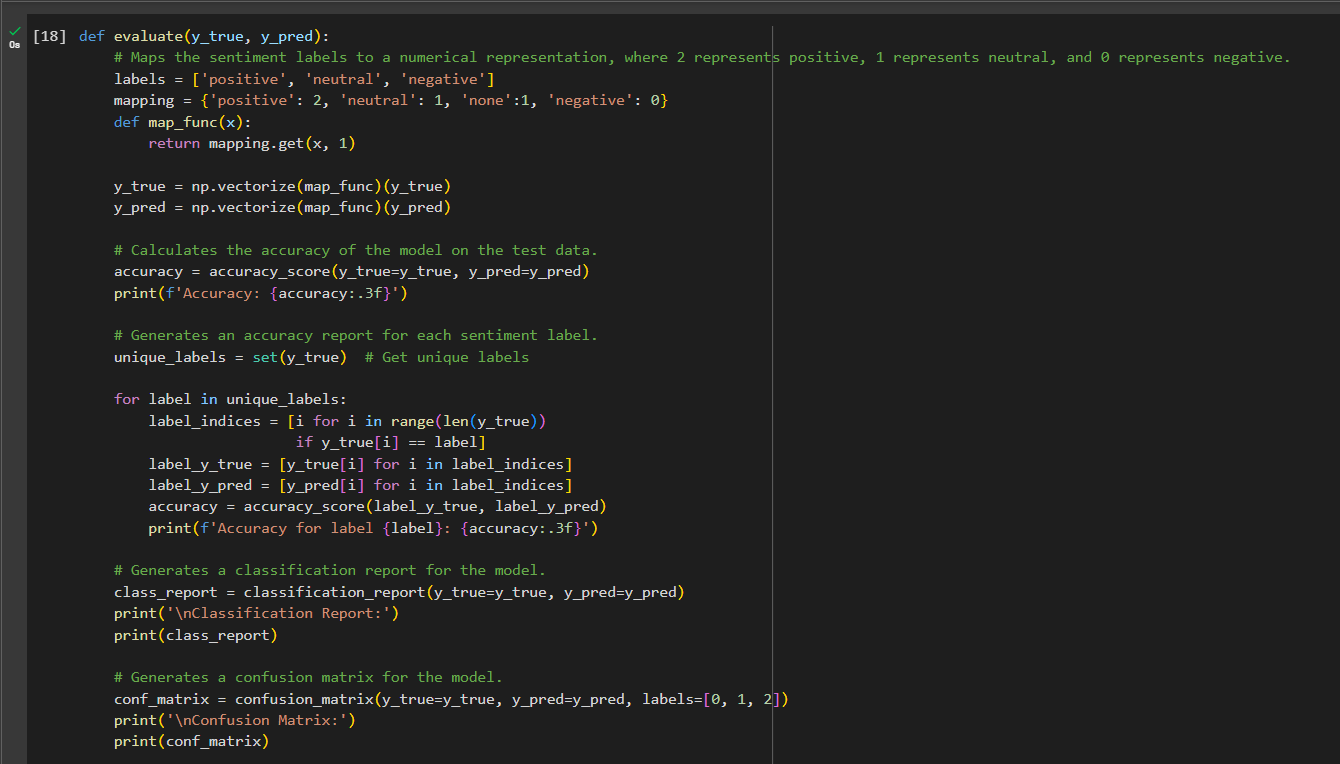






Next we create a function to evaluate the results from our fine-tuned sentiment model. The function performs the following steps:

* Maps the sentiment labels to a numerical representation, where 2 represents positive, 1 represents neutral, and 0 represents negative.
* Calculates the accuracy of the model on the test data.
* Generates an accuracy report for each sentiment label.
* Generates a classification report for the model.
* Generates a confusion matrix for the model.



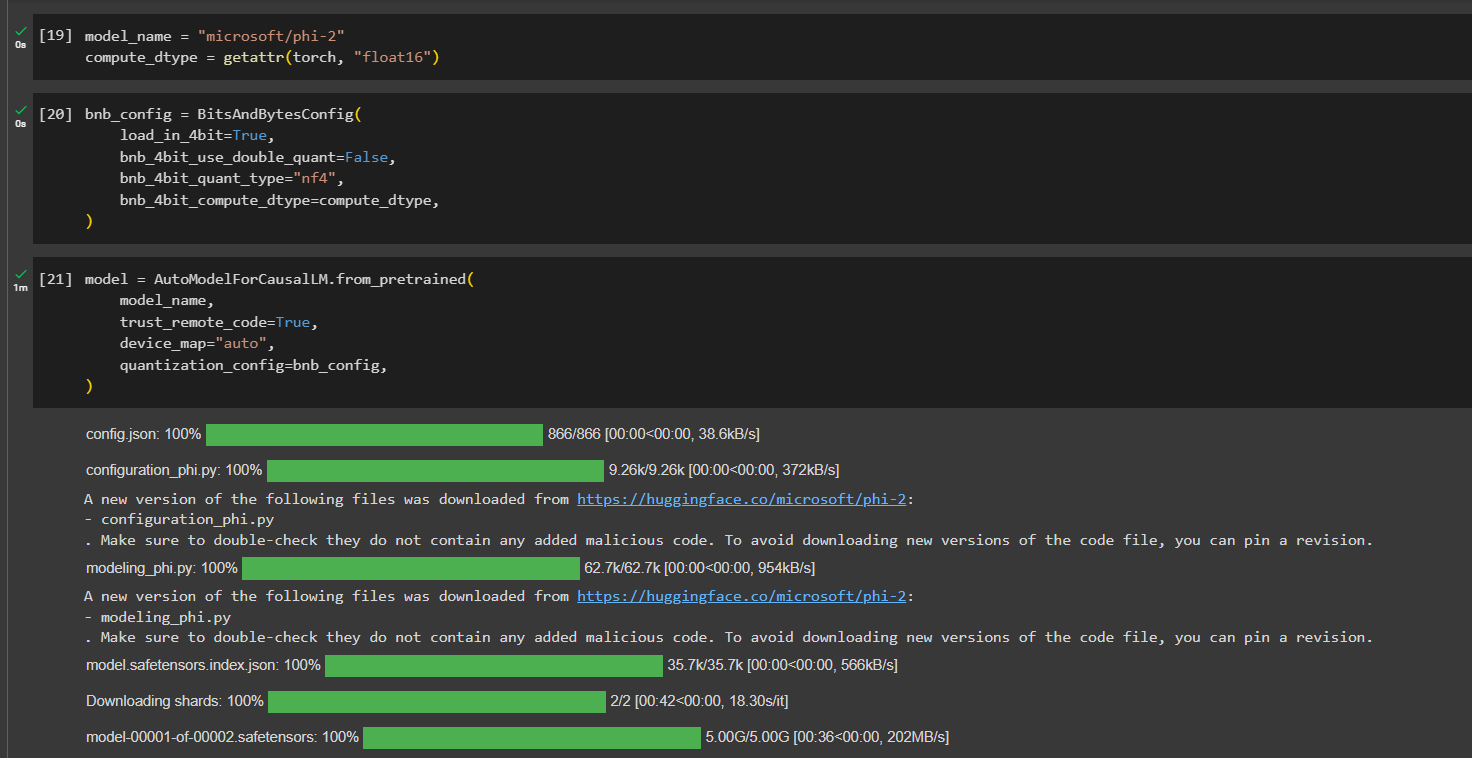
Next we need to take care of the model, which is a 7b-v0.1-hf (7 billion parameters, version 0.1, in the HuggingFace compatible format), loading from Kaggle models and quantization.

Model loading and quantization:

* First the code loads the Phi-2 language model from the Hugging Face Hub.
* Then the code gets the float16 data type from the torch library. This is the data type that will be used for the computations.
* Next, it creates a BitsAndBytesConfig object with the following settings:
  1. load\_in\_4bit: Load the model weights in 4-bit format.
  2. bnb\_4bit\_quant\_type: Use the "nf4" quantization type. 4-bit NormalFloat (NF4), is a new data type that is information theoretically optimal for normally distributed weights.
  3. bnb\_4bit\_compute\_dtype: Use the float16 data type for computations.
  4. bnb\_4bit\_use\_double\_quant: Do not use double quantization (reduces the average memory footprint by quantizing also the quantization constants and saves an additional 0.4 bits per parameter.).
* Then the code creates a AutoModelForCausalLM object from the pre-trained Phi-2 language model, using the BitsAndBytesConfig object for quantization.
* After that, the code disables caching for the model.
* Finally the code sets the pre-training token probability to 1.

Tokenizer loading:

* First, the code loads the tokenizer for the Phi-2 language model.
* Then it sets the padding token to be the end-of-sequence (EOS) token.
* Finally, the code sets the padding side to be "left", which means that the input sequences will be padded on the left side.





**Using the base Phi-2 model for Sentiment Analysis**

In the next cell, we set a function for predicting the sentiment of a news headline using the Phi-2 language model. The function takes three arguments:

test: A Pandas DataFrame containing the news headlines to be predicted. model: The pre-trained Phi-2 language model. tokenizer: The tokenizer for the Phi-2 language model.

The function works as follows:

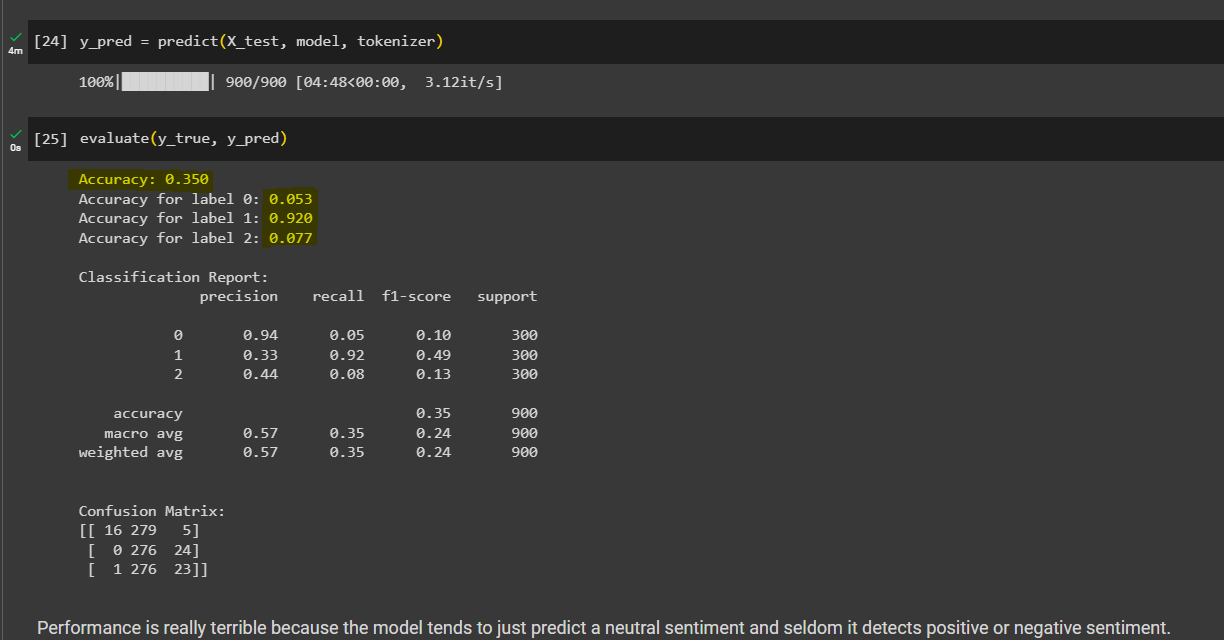
1. For each news headline in the test DataFrame:
   * Create a prompt for the language model, which asks it to analyze the sentiment of the news headline and return the corresponding sentiment label.
   * Use the pipeline() function from the Hugging Face Transformers library to generate text from the language model, using the prompt.
   * Extract the predicted sentiment label from the generated text.
   * Append the predicted sentiment label to the y\_pred list.
2. Return the y\_pred list.

The pipeline() function from the Hugging Face Transformers library is used to generate text from the language model. The task argument specifies that the task is text generation. The model and tokenizer arguments specify the pre-trained Phi-2 language model and the tokenizer for the language model. The max\_new\_tokens argument specifies the maximum number of new tokens to generate. The temperature argument controls the randomness of the generated text. A lower temperature will produce more predictable text, while a higher temperature will produce more creative and unexpected text.

The if statement checks if the generated text contains the word "positive". If it does, then the predicted sentiment label is "positive". Otherwise, the if statement checks if the generated text contains the word "negative". If it does, then the predicted sentiment label is "negative". Otherwise, the if statement checks if the generated text contains the word "neutral". If it does, then the predicted sentiment label is "neutral.



**Evaluating Base Model Performance**



Poor performance of base model.

**Fine Tuning!**

In the next cell we set everything ready for the fine-tuning. We configures and initializes a Simple Fine-tuning Trainer (SFTTrainer) for training a large language model using the Parameter-Efficient Fine-Tuning (PEFT) method, which should save time as it operates on a reduced number of parameters compared to the model's overall size. The PEFT method focuses on refining a limited set of (additional) model parameters, while keeping the majority of the pre-trained LLM parameters fixed. This significantly reduces both computational and storage expenses. Additionally, this strategy addresses the challenge of catastrophic forgetting, which often occurs during the complete fine-tuning of LLMs.

PEFTConfig:

The peft\_config object specifies the parameters for PEFT. The following are some of the most important parameters:

* lora\_alpha: The learning rate for the LoRA update matrices.
* target\_modules: layers of the model that are targeted to improve performance.
* lora\_dropout: The dropout probability for the LoRA update matrices.
* r: The rank of the LoRA update matrices.
* bias: The type of bias to use. The possible values are none, additive, and learned.
* task\_type: The type of task that the model is being trained for. The possible values are CAUSAL\_LM and MASKED\_LM.

TrainingArguments:

The training\_arguments object specifies the parameters for training the model. The following are some of the most important parameters:

* output\_dir: The directory where the training logs and checkpoints will be saved.
* num\_train\_epochs: The number of epochs to train the model for.
* per\_device\_train\_batch\_size: The number of samples in each batch on each device.
* gradient\_accumulation\_steps: The number of batches to accumulate gradients before updating the model parameters.
* optim: The optimizer to use for training the model.
* save\_steps: The number of steps after which to save a checkpoint.
* logging\_steps: The number of steps after which to log the training metrics.
* learning\_rate: The learning rate for the optimizer.
* weight\_decay: The weight decay parameter for the optimizer.
* fp16: Whether to use 16-bit floating-point precision.
* bf16: Whether to use BFloat16 precision.
* max\_grad\_norm: The maximum gradient norm.
* max\_steps: The maximum number of steps to train the model for.
* warmup\_ratio: The proportion of the training steps to use for warming up the learning rate.
* group\_by\_length: Whether to group the training samples by length.
* lr\_scheduler\_type: The type of learning rate scheduler to use.
* report\_to: The tools to report the training metrics to.
* evaluation\_strategy: The strategy for evaluating the model during training.

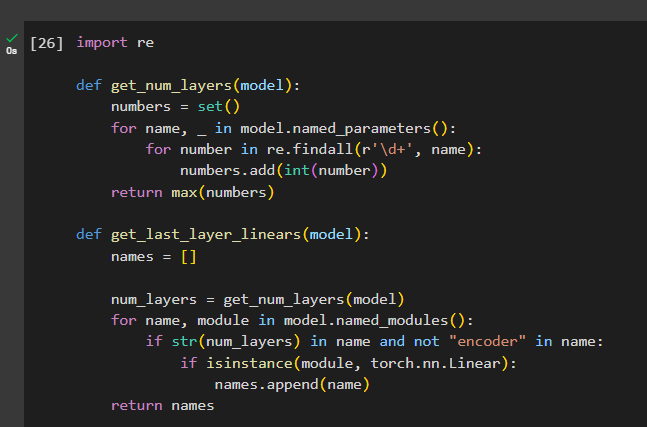
SFTTrainer:

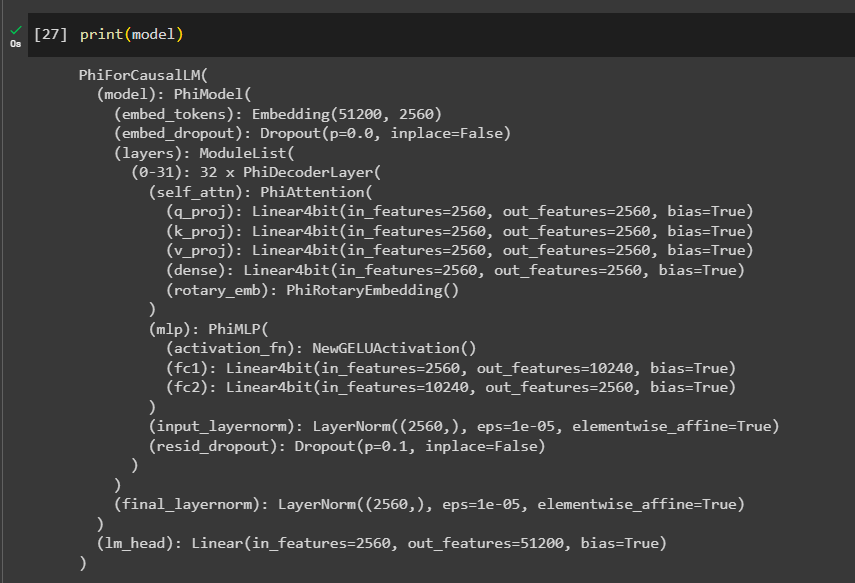
The SFTTrainer is a custom trainer class from the PEFT library. It is used to train large language models using the PEFT method.

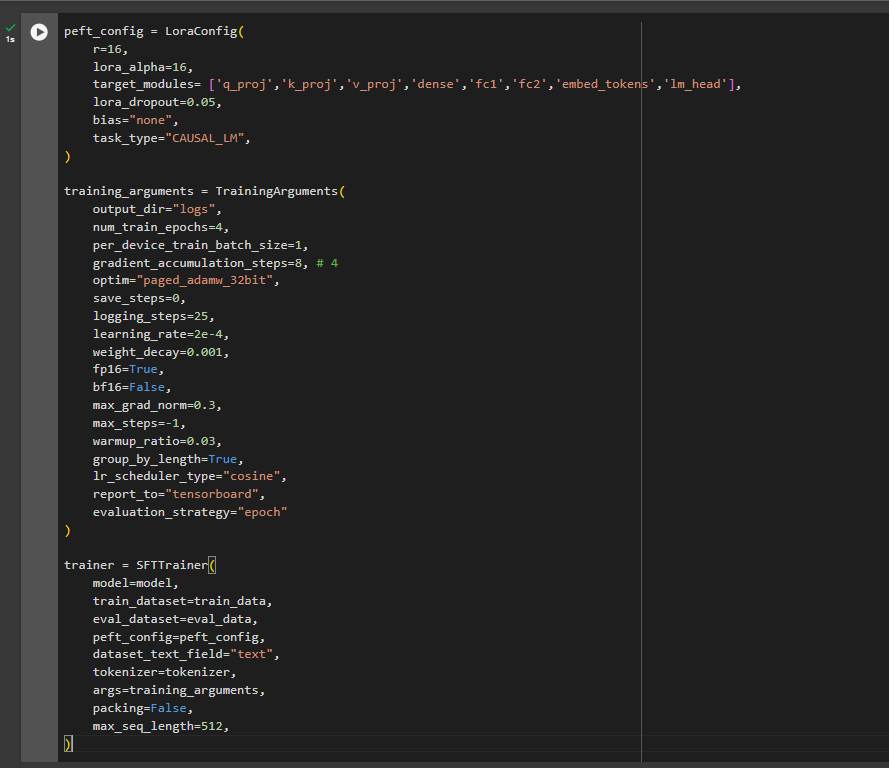
The SFTTrainer object is initialized with the following arguments:

* model: The model to be trained.
* train\_dataset: The training dataset.
* eval\_dataset: The evaluation dataset.
* peft\_config: The PEFT configuration.
* dataset\_text\_field: The name of the text field in the dataset.
* tokenizer: The tokenizer to use.
* args: The training arguments.
* packing: Whether to pack the training samples.
* max\_seq\_length: The maximum sequence length.

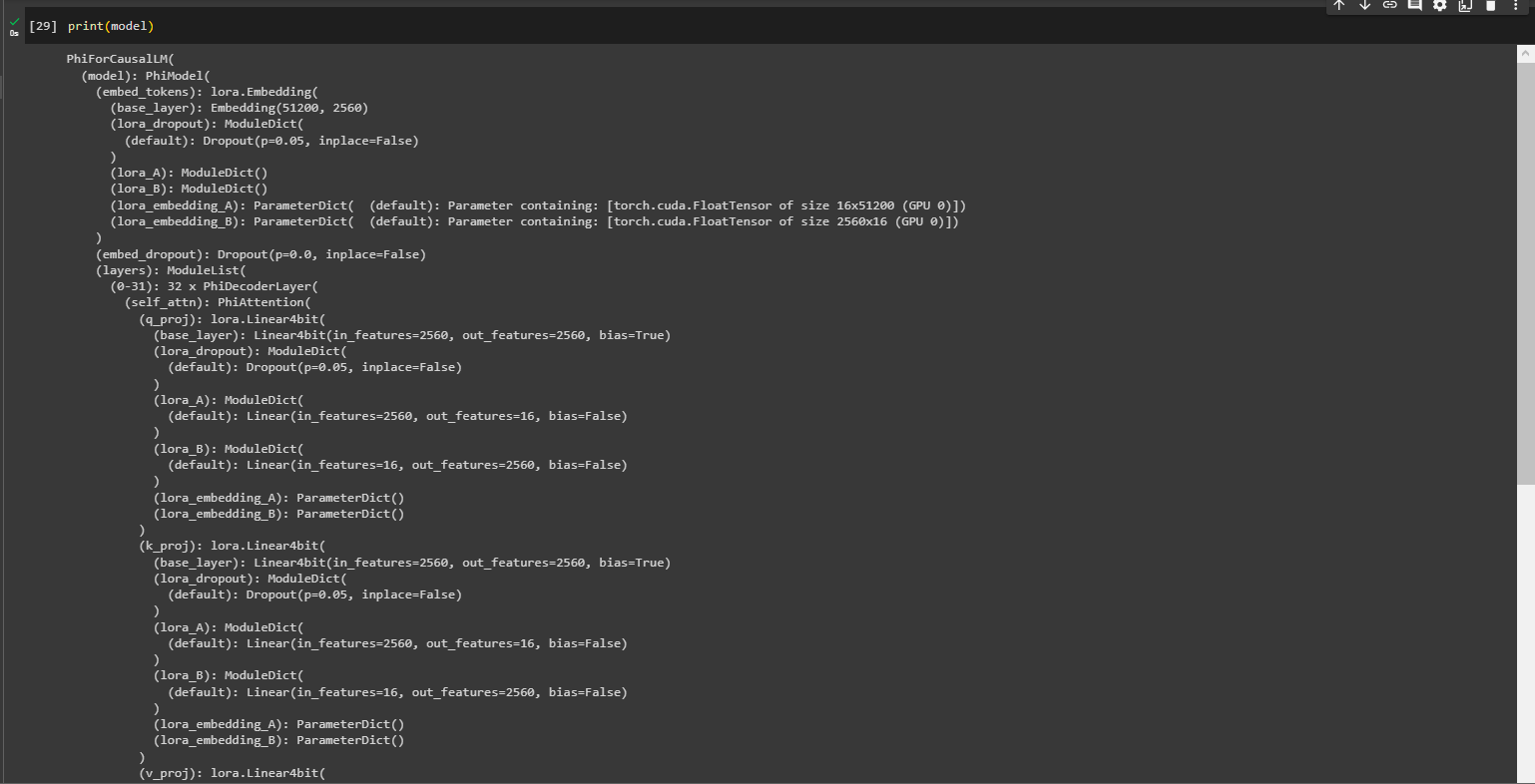
Once the SFTTrainer object is initialized, it can be used to train the model by calling the train() method

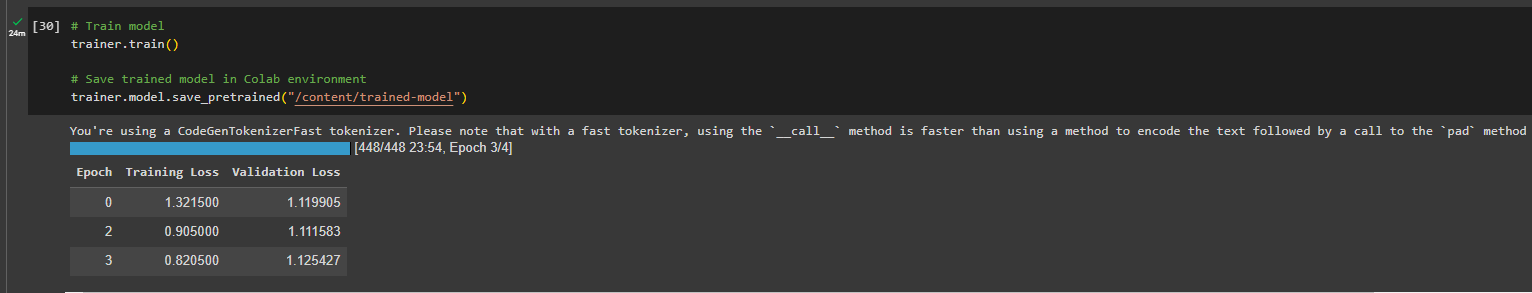






target\_modules= ['q\_proj','k\_proj','v\_proj','dense','fc1','fc2','embed\_tokens','lm\_head']. Here all the possible layers that can be targeted, are targeted.





| **Epoch** | **Training Loss** | **Validation Loss** |
| --- | --- | --- |
| 0 | 1.321500 | 1.119905 |
| 2 | 0.905000 | 1.111583 |
| 3 | 0.820500 | 1.125427 |

The above shows the decrease in training loss over epochs but epoch 3 shows slight increase in validation loss.

**Epoch:** An epoch is one complete pass through the entire training dataset. In each epoch, the model is trained on the entire dataset, and the training process is repeated for the specified number of epochs.

**Training Loss:** The training loss is a measure of how well the model is performing on the training dataset. It represents the error between the model's predictions and the actual target values during training. The goal is to minimize this loss. A decreasing training loss indicates that the model is learning from the data.

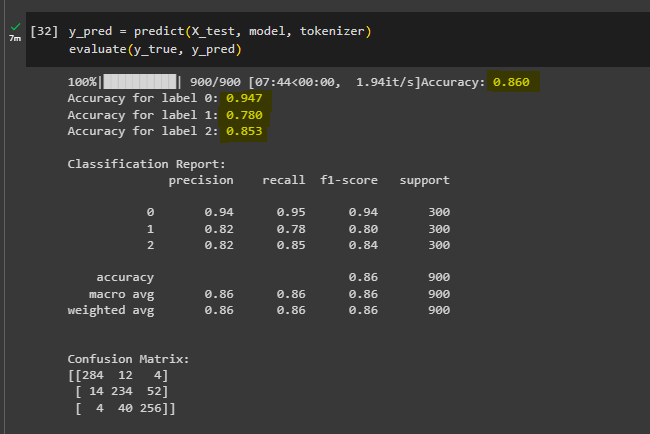
**Validation Loss:** The validation loss is a measure of how well the model is performing on a separate validation dataset that it hasn't seen during training. It serves as an estimate of the model's generalization performance on new, unseen data. Like the training loss, the goal is to minimize the validation loss. An increasing validation loss may indicate overfitting, where the model is too specialized to the training data and performs poorly on new data.

**Epoch 0: The initial state of the model. The training loss is 1.321500, and the validation loss is 1.119905.**

**Epoch 1: After the first pass through the training dataset, the training loss decreases to 0.905000, but the validation loss slightly increases to 1.111583. This could be an early sign of overfitting, as the model might be fitting the training data too closely and not generalizing well to new data.**

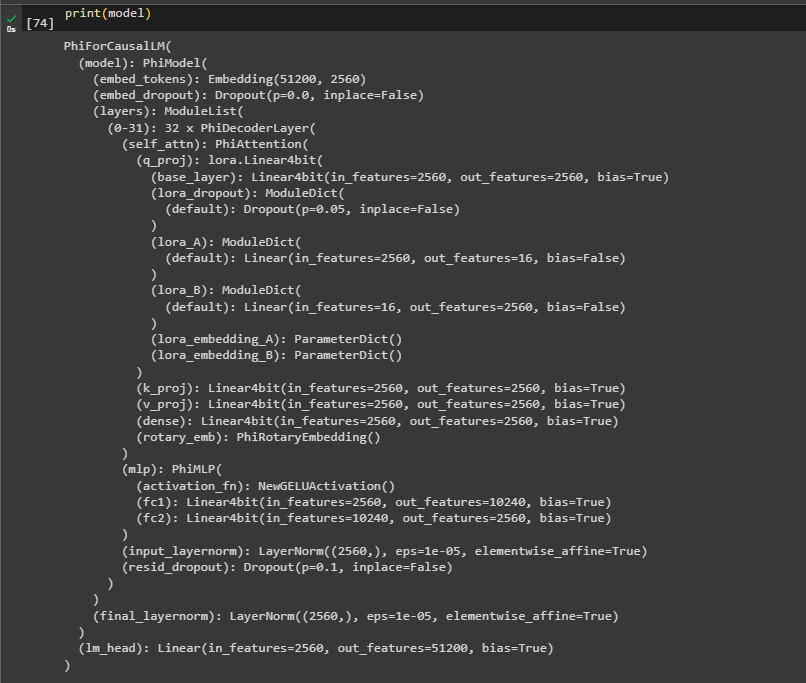
**Epoch 2: The training loss continues to decrease to 0.820500, but the validation loss increases further to 1.125427. This increase in validation loss suggests that the model's performance on unseen data is degrading, which could be a more pronounced sign of overfitting.**

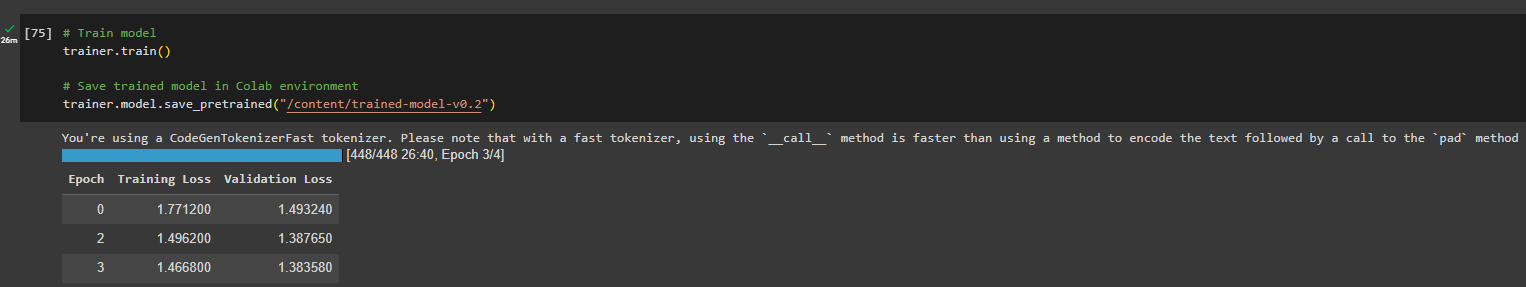
After fine tuning, we see evaluation as -

**  
  
We see a very significant and noticeable jump in accuracy of the model after fine-tuning it.**

**Using different target modules for Fine Tuning**

target\_modules=["Wqkv", "q\_proj"]





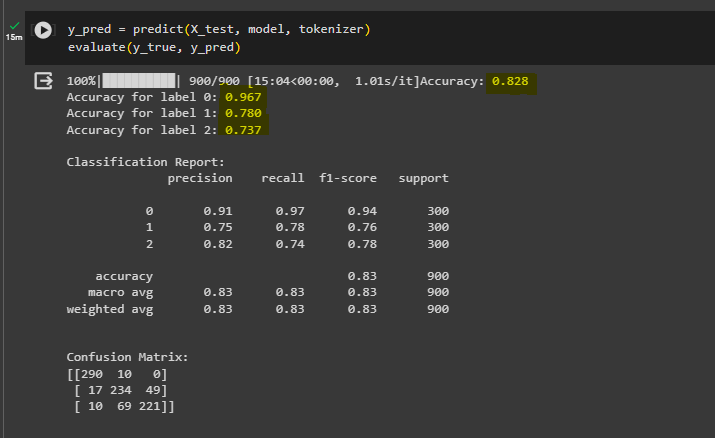
| **Epoch** | **Training Loss** | **Validation Loss** |
| --- | --- | --- |
| 0 | 1.771200 | 1.493240 |
| 2 | 1.496200 | 1.387650 |
| 3 | 1.466800 | 1.383580 |

**Epoch 0: The initial state of the model. The training loss is 1.771200, and the validation loss is 1.493240.**

**Epoch 2: After the first two passes through the training dataset, the training loss decreases to 1.496200. The validation loss also decreases to 1.387650. This suggests that the model is learning and improving its performance on both the training and validation datasets.**

**Epoch 3: The training loss continues to decrease to 1.466800, and the validation loss remains relatively stable at 1.383580. This indicates that the model is still improving on the training data, and its performance on the validation data is consistent.**

The model is showing a decrease in both training and validation losses, which is a positive sign. It suggests that the model is learning and generalizing well to unseen data. Monitoring for overfitting is always important, but based on the provided results, the model seems to be making progress in a positive direction. If these trends continue, it indicates that the training process is effective.

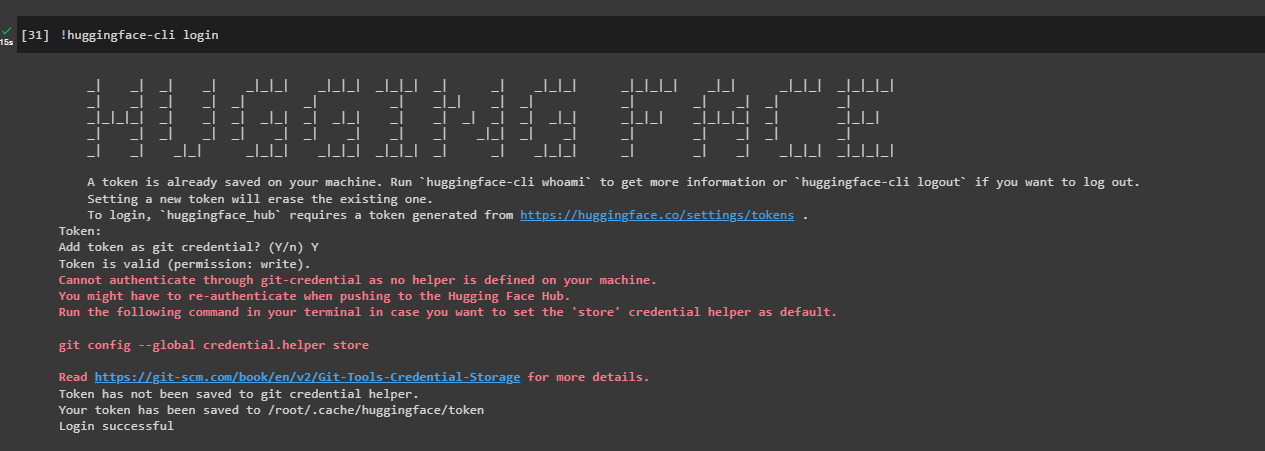


The accuracy of the model has not significantly decreased but overfitting is avoided by changing the target modules in use with PEFT/LoRA.

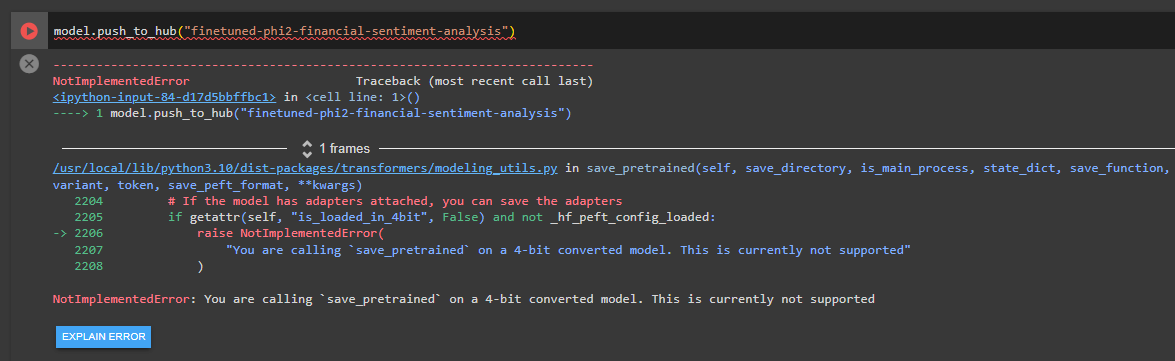
**The model is now fine tuned to the application of providing sentiments for financial application.**

**Pushing the model to Hugging Face**

Login to hugging face in the terminal (! for cli command)



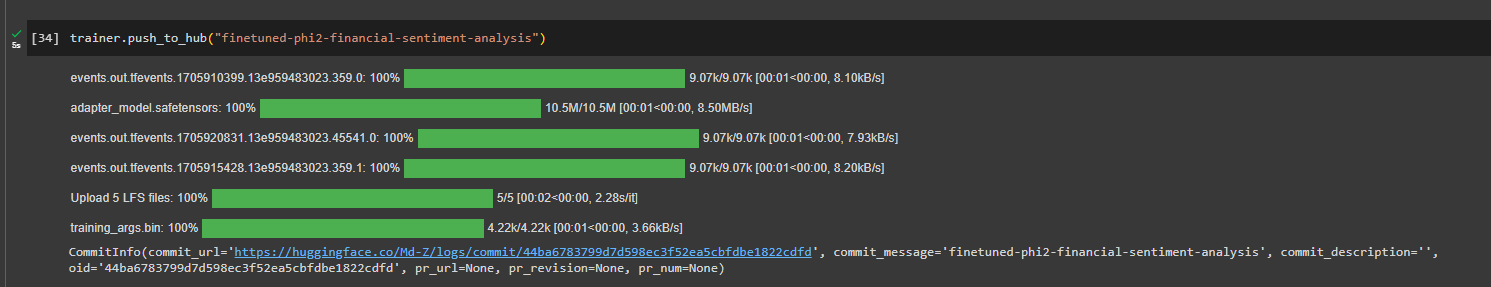
Possible error when pushing the model



This error is due to unavailable support for 4-bit models to be pushed to the hub in the current version of the transformer library.   
  
Solution -   
Install the latest dev version of the library

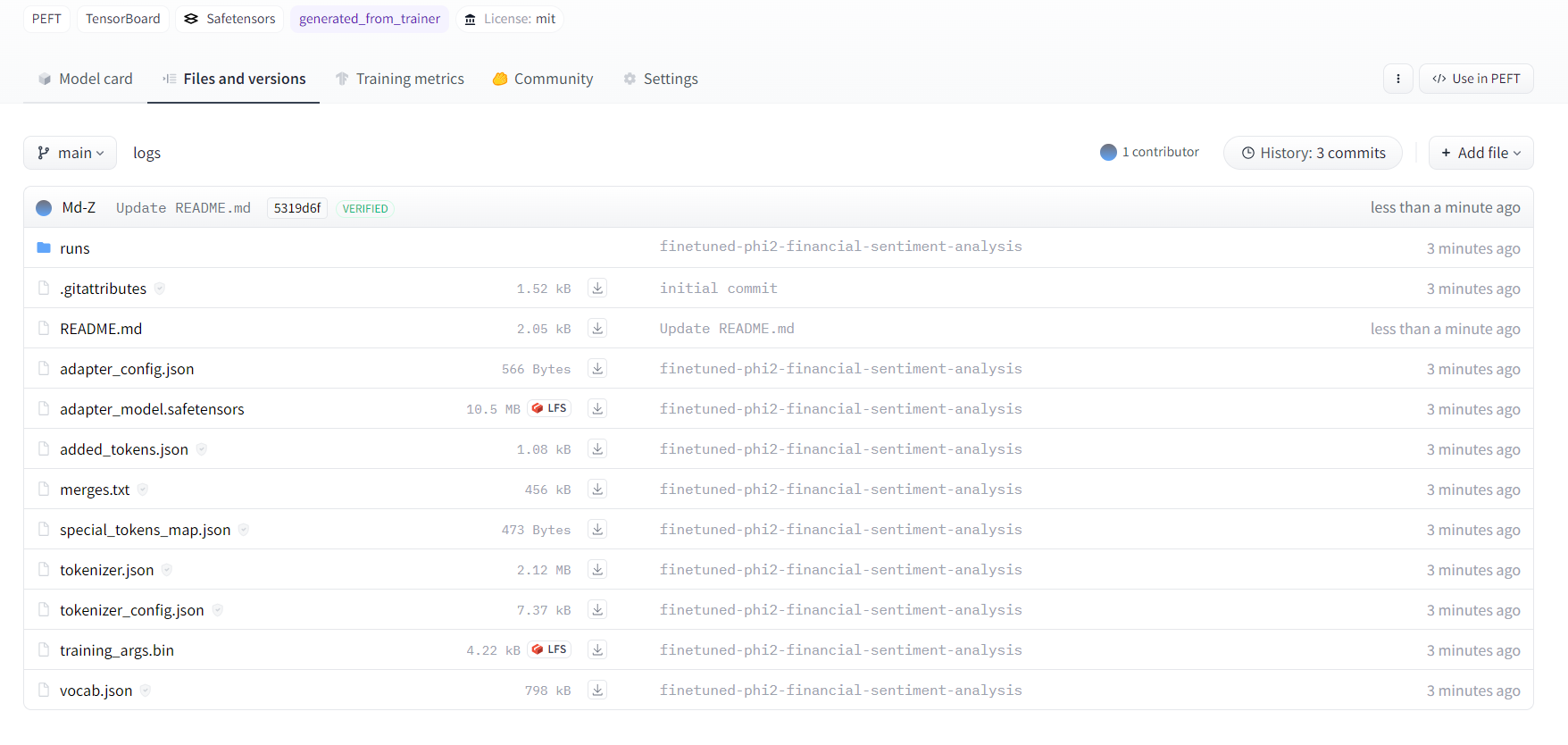


Now, to push the model to a hugging face repository, use code -

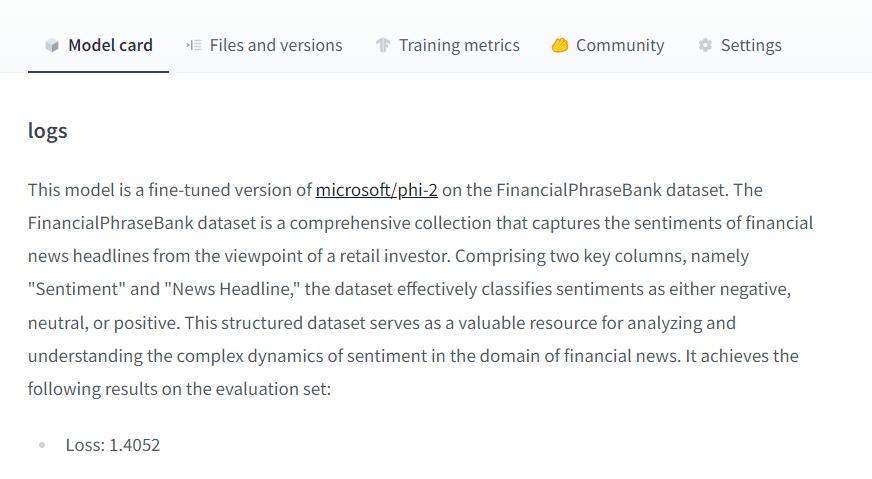


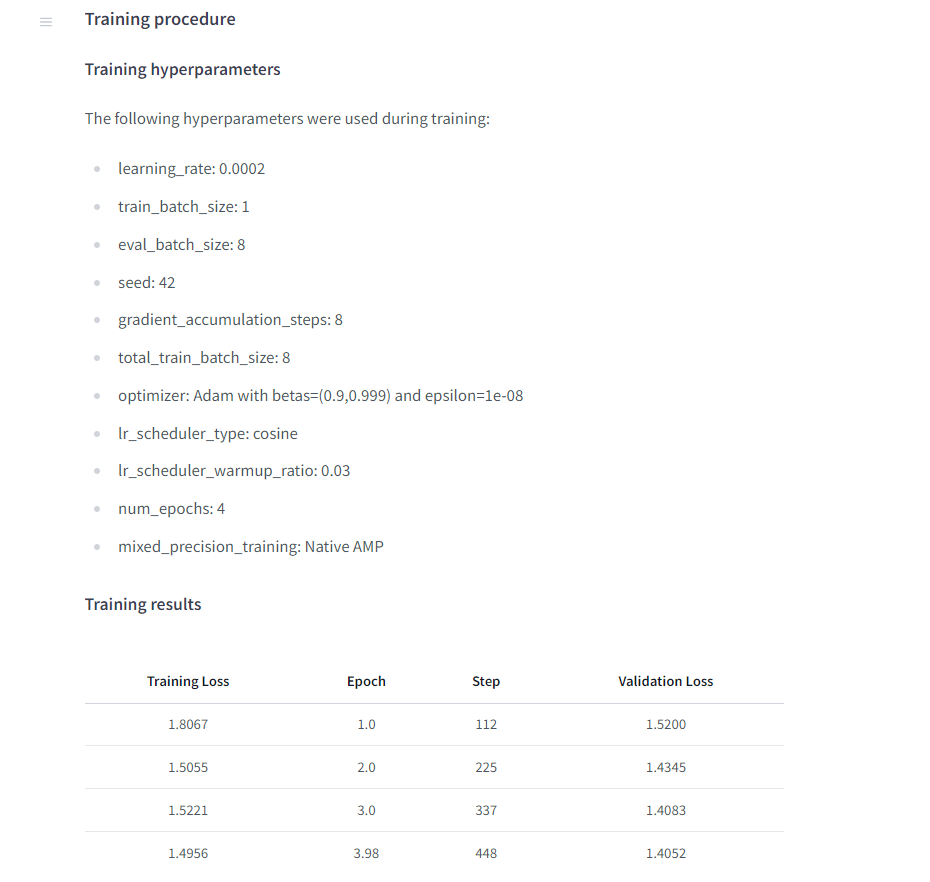
trainer.push\_to\_hub("finetuned-phi2-financial-sentiment-analysis")

Clicking on the link to the commit, we can see the model and generated model card for this fine tuned model.

****

**—----------------------------------------------------------------------------------------------------------------------------**

****

****