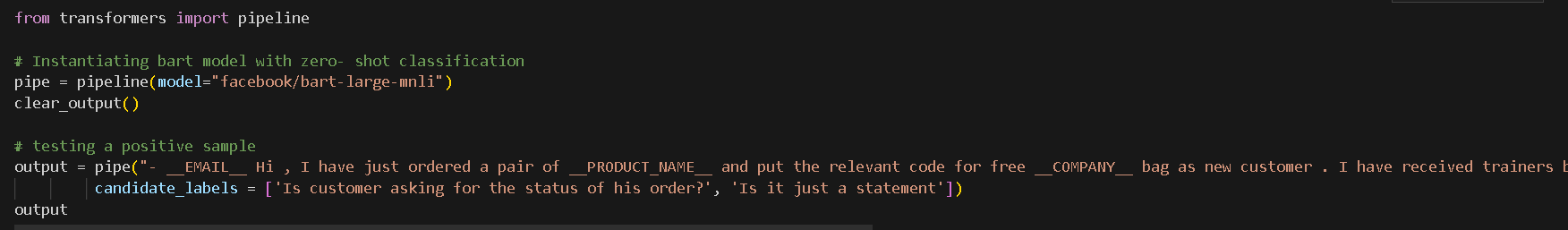
This code uses the Hugging Face transformers library to create a text classification pipeline using the BART (Bidirectional and Auto-Regressive Transformers) model, specifically the facebook/bart-large-mnli pre-trained model. BART is a transformer-based model that is often used for various natural language processing tasks.

Let's break down the code step by step:



1. **Importing the necessary libraries:**

from transformers import pipeline

This line imports the pipeline class from the transformer’s library. The pipeline class simplifies the process of using pre-trained models for various natural language processing tasks.

1. **Instantiating the BART model for zero-shot classification:**

pipe = pipeline(model="facebook/bart-large-mnli")

Here, a new pipeline is created using the **BART model for zero-shot classification**. The model used is **facebook/bart-large-mnli**, which is pre-trained on the **MNLI** (**MultiNLI**) dataset for natural language inference tasks. pipeline automatically handles loading the pre-trained model and setting it up for **zero-shot classification**.

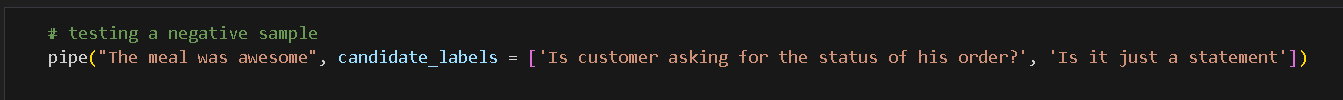
1. **Testing a positive sample:**

**output = pipe("- \_\_EMAIL\_\_ Hi , I have just ordered a pair of \_\_PRODUCT\_NAME\_\_ and put the relevant code for free \_\_COMPANY\_\_ bag as new customer . I have received trainers but no bag . Will this be sent separately? \n Seems to be an inefficient system, or was this an oversight?\n Kind regards\n \_\_NAME\_\_",**

**candidate\_labels=['Is customer asking for the status of his order?', 'Is it just a statement'])**

This code snippet uses the instantiated pipeline (**pipe**) to perform **zero-shot classification** on a sample text. The sample text appears to be a **customer inquiry** about a missing bag in their order. The **candidate\_labels** parameter specifies the possible labels or categories for classification. In this case, the two candidate labels are:

* **'Is customer asking for the status of his order?'**
* **'Is it just a statement'**

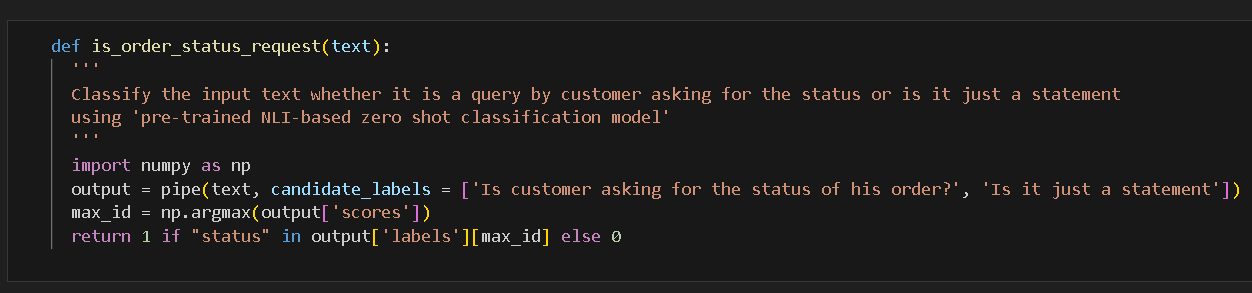
****

1. This code is a continuation of the previous code and is testing a negative sample using the same zero-shot classification pipeline created with the BART model. Let's break down the code:

The text to be classified is "The meal was awesome." This is a positive statement about a meal, and the goal is to see how well the model can correctly classify this positive statement into the specified candidate labels.

The candidate labels are the same as in the previous example:

* **'Is customer asking for the status of his order?'**
* **'Is it just a statement'**

****

**Importing Libraries:**

The function imports the NumPy library as np. NumPy is commonly used for numerical operations in Python.

**Zero-Shot Classification:**

The function uses the pipe object (presumably created earlier with a pre-trained NLI-based zero-shot classification model) to classify the input text. The candidate labels are specified as whether the customer is asking for the status of their order or if it's just a statement.

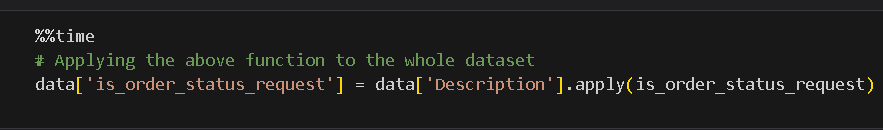
**Identifying Maximum Score Index:**

The function uses NumPy's argmax function to find the index with the maximum score in the array of scores returned by the model. This index corresponds to the predicted label with the highest confidence.

**Checking for "status" in Predicted Label:**

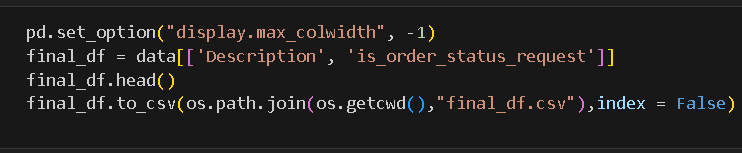
The function returns 1 if the word "status" is present in the predicted label associated with the maximum confidence score, and 0 otherwise. This is a simple way to determine if the model has classified the input text as a query about the order status.

In summary, this function provides a convenient way to determine whether a given text input is likely a customer query about the status of their order or just a statement, using a pre-trained NLI-based zero-shot classification model. The function returns 1 if the word "status" is present in the predicted label with the highest confidence score and 0 otherwise.

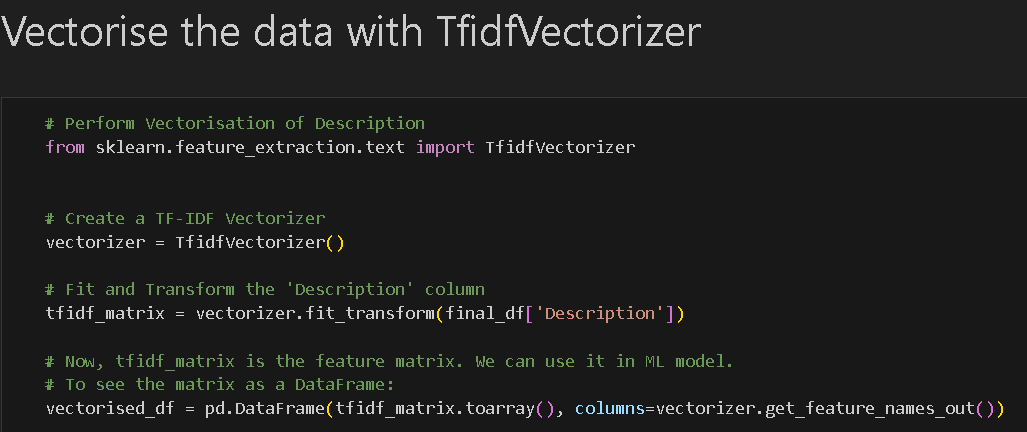


this code is using the apply function to classify each description in the **'Description**' column of the DataFrame data using the **is\_order\_status\_request** function. The results of these classifications are stored in a new column named **'is\_order\_status\_request'**. This is a common pattern when working with pandas DataFrames to apply a function to each element in a column **and create a new column with the results**.

This step took around **1.2 hours in local** but around **25 mins in colab pro** with **v100 gpu.**



this code sets display options for pandas, creates a subset **DataFrame (final\_df)** by selecting specific columns from the original DataFrame (data), displays the first few rows of the new DataFrame, and then saves the DataFrame to a CSV file named "**final\_df.csv**" in the current working directory.



This code performs text vectorization using the TF-IDF (Term Frequency-Inverse Document Frequency) technique. TF-IDF is commonly used in natural language processing to convert a collection of text documents into numerical feature vectors.

the **TfidfVectorizer** from scikit-learn to convert the 'Description' column of the DataFrame (final\_df) into a TF-IDF feature matrix (**tfidf\_matrix**). The resulting matrix is then converted into a DataFrame (**vectorised\_df**) for further analysis or for use as input features in a machine learning model.

This step took less than 153 **millisecs in colab pro** with **v100 gpu** but takes few hours in local.