**Task 1: Deep Learning-Based Name Matching**

**Choosing a BERT model:**

Word Piece Tokenization: BERT uses a subword tokenization approach (WordPiece), which means it can handle out-of-vocabulary words and break down complex words into smaller meaningful subunits. This is particularly useful when dealing with abbreviated or truncated names commonly found in transaction descriptions.

Pre-trained Language Model: BERT is pre-trained on a large corpus of text data, which includes a wide range of language patterns and nuances. This pre-training helps BERT to generalize well across different domains and tasks, including matching names with varying forms.

Transfer Learning Benefits: Leveraging BERT for name matching involves transfer learning, where the model's pre-trained knowledge is transferred to a specific task (name matching). This often leads to improved performance with less labeled data required for training -- in our case, we don't have any data

State-of-the-art Performance: BERT has demonstrated state-of-the-art performance on various NLP benchmarks and tasks. It's a widely adopted and well-tested model that can provide strong performance for name matching task.

**Model Architecture:**

Token Embedding Layer:

- Convert input names and transaction descriptions into token sequences.

- Use a tokenization method that captures subword units and handles variations/abbreviations effectively.

Pre-trained Language Model (BERT):

- Fine-tune a pre-trained transformer model (BERT) for the name matching task:

- Input Representation\*\*: Convert tokenized inputs into contextualized embeddings using BERT's token embedding layer.

Sequence Matching Layer:

- BERT-based Approach:

- Use BERT's output embeddings (token representation) as input to downstream sequence matching layers (fully connected layers, softmax for classification).

Output Layer:

- Output layer to predict the similarity or matching score between customer names and transaction descriptions:

- Ranking/Scoring: Assign a similarity score (cosine similarity) to quantify the match strength between names and transaction descriptions.

I have a jupyter notebook for the whole task implementation and a more descriptive explanation for all steps.

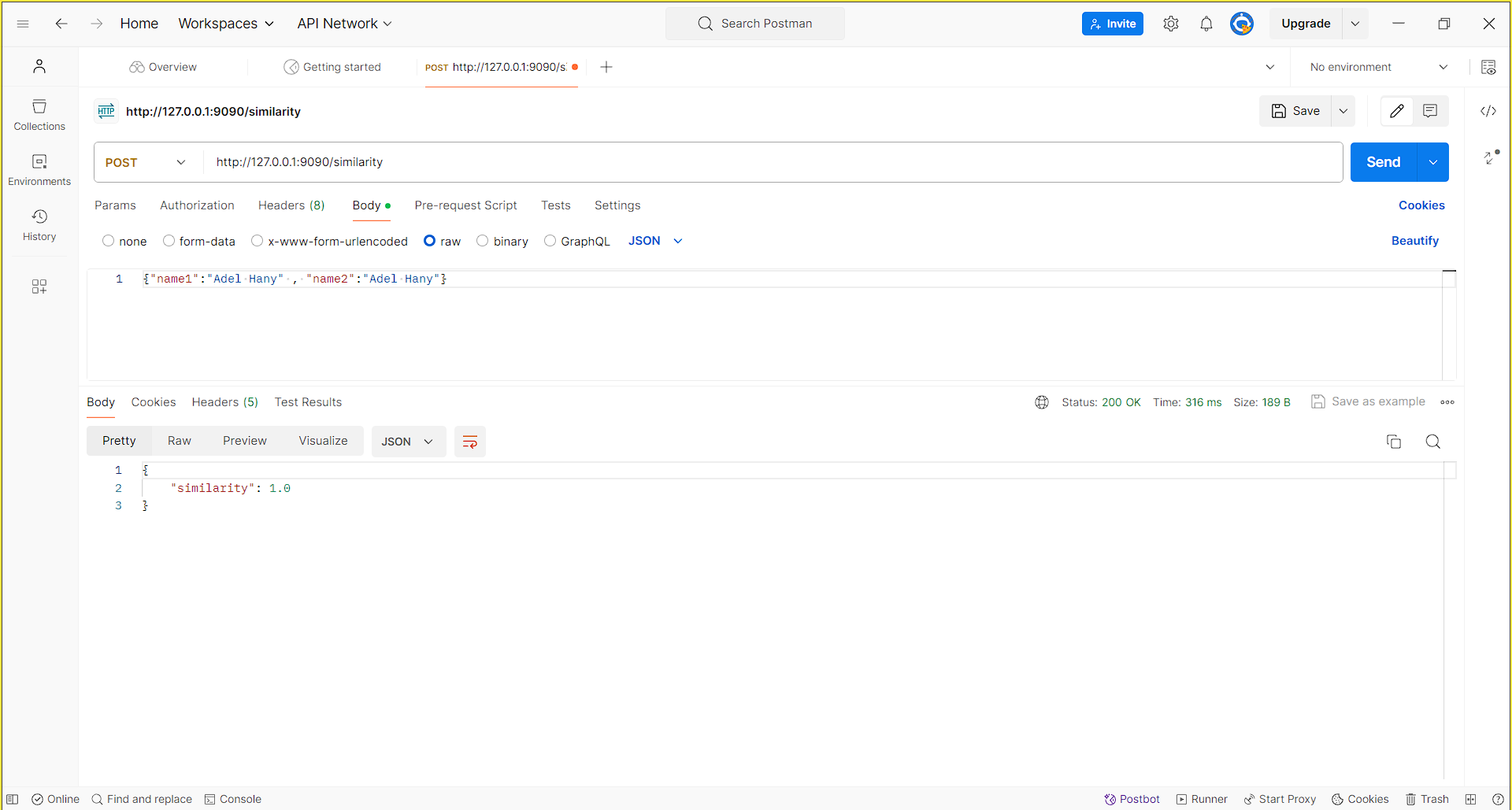
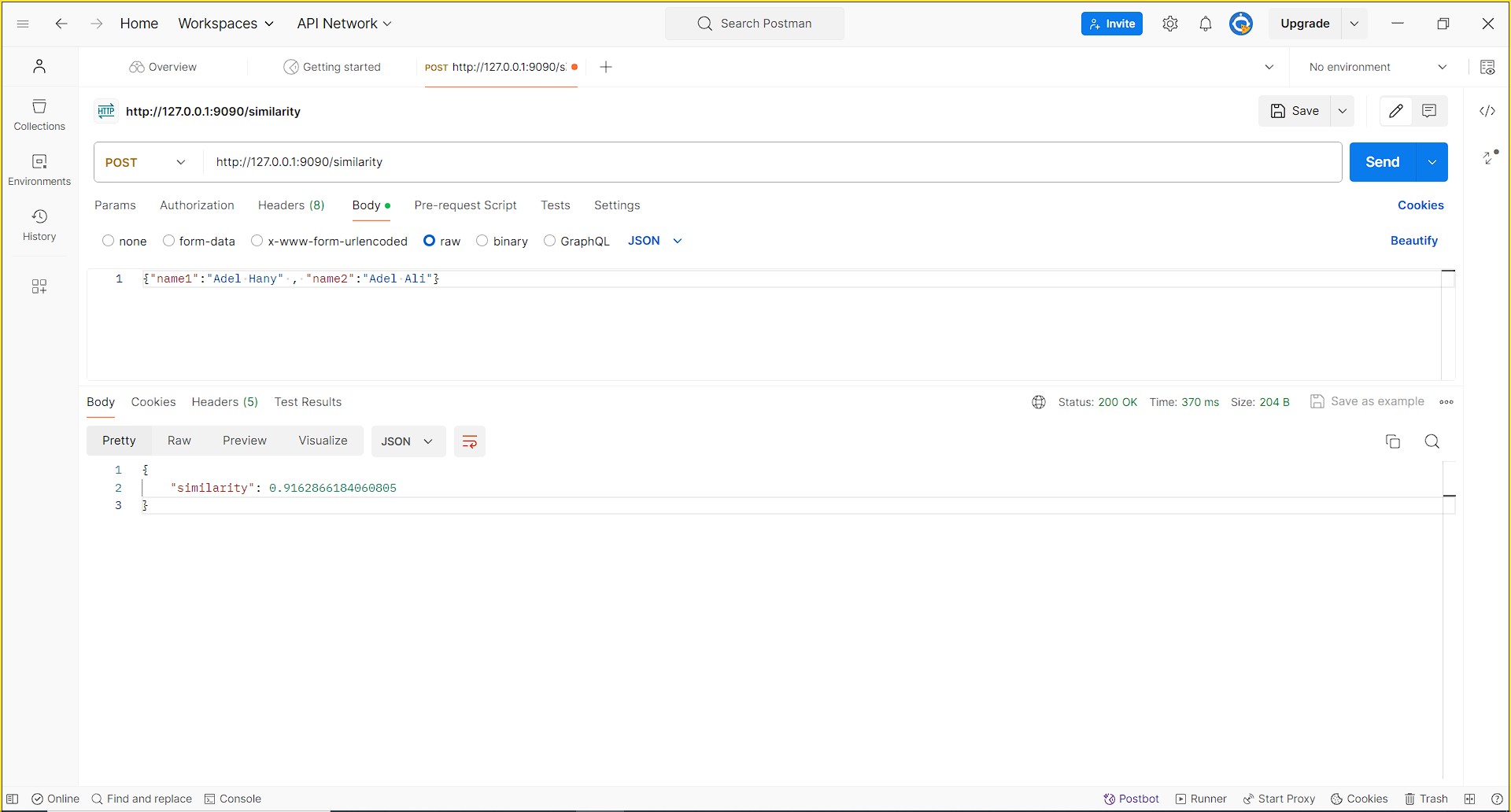
From this notebook, I made my Flask application and tested it through postman.

Open the file (app.py) through VS Code and run “python app.py” through terminal.

Then open postman and navigate to the HTTP link. Make the method: post.

Body > raw > and input a dictionary with 2 names to test them and get the similarity score.

For ex:  
{"name1":"Adel Hany" , "name2":"Hany Adel"}



I also provided an mp4 video for the testing.

Thank you