Q1) Creating a rudimentary Question-Answering system with the help of fine-tuned BERT model.

BERT stands for Bidirectional Encoder Representations from Transformers. It leverages a multi-layered Transformer encoder architecture. Unlike unidirectional models, transformers can analyze all words simultaneously through a mechanism called "self-attention."

Below I provide a short summary of how I have trained BERT on CoQA dataset.

CoQA is a Conversational Question Answering dataset released by Stanford NLP in 2019. It is a large-scale dataset for building Conversational Question Answering Systems. This dataset aims to measure the ability of machines to understand a text passage and answer a series of interconnected questions that appear in a conversation.

Summary:
1) Firstly I import the necessary libraries and remove the unwanted column that was not required.

2) The cleaned data is then processed has 3 columns namely "text","question" and "answer"

3) For our task, we will use the BertForQuestionAnswering class from the transformers library which is fine-tuned on the SQuAD Benchmark.

4) Once we import the latter mentioned library we then tokenize question and text as a pair and save them as "input_ids".

5) BERT has a unique way of processing tokenized inputs. There are 2 main token namely [CLS] and [SEP]. [CLS] stands for "classification" is used to represent sentence level classification while [SEP] token is used to Separate to sentences.

6) Then our task was to create segment IDs . In essence, we prepare the input for a model that expects two segments of text, differentiating between the question and the text passage it needs to analyse.

7) The main part of this task – Extracting answers from the model's output:

• output.start_logits and output.end_logits → These are tensors containing scores for every token in the input text passage. These scores indicate how likely a word is the beginning (start) or the end of the answer.

• torch.argmax(output.start_logits)→This line uses the argmax() function from the PyTorch library to find the index of the element in start_logits with the highest score. This signifies the most likely starting position of the answer within the tokens.
• torch.argmax(output.end_logits)→ This does the similar things as the above mentioned part but for the end position of the answer within the tokens.

After this is done, we put a conditional block to check whether end position comes after the start position which ensures that the correct answer is extracted .

If the positions are valid, we extract the tokens from the original tokens list based on start and end indices and then return a string of answer.

8) We then ask questions from the given JSON data file and we see that it performs cool.