1) **Code Overview** (*An overview of the function of the code (i.e., what it does and what it can be used for).*)

Our project was to build a web application that took in as an input meeting transcript data and output the topics that were discussed in this meeting. With the proliferation of online meeting data we think there is useful data for an organization to gain insights into their organizations from their meeting transcripts now that they are mostly virtual meetings. We built a program that analyzes meeting transcripts and weights which topics were discussed in the meeting. A user can go to our website, input the text string into the input module and our website will return the weights of how much each topic was discussed. We spent focused on the model development more than the user interface but we think this can easily be extended to an application that programmatically aggregates topics across many meetings to see what is trending within your organization which would be useful information for a executive and management teams.

2) **Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.**

There are three main components of this project: 1) user interface 2) model and 3) model training. This will provide an overview of the code structure eand implementation but the detailed notes are commented on in the code.

The user interface is a web application hosted on the streamlit service. Our user interface is in the app.py file. The web page displays to the user a list of the topics we will be selecting from at the top of the webpage. The user then inputs a text string of their preferred meeting transcript into the web dialogue. The web service takes this text string and calls the model to get the weights for each topic. The web app displays the weights of each topic to the user.

The model we choose is the latent Dirichlet allocation (LDA) model. We experimented with Latent Semantic Analysis, LDA and PLSA models. We ran experiments by running a transcript through the models and manually analyzing the results and found that the LDA model was the most accurate. In the LDA model the programmer needs to select the number of topics there will be. We selected three topics because this would be a useful number for the user. The model will then look have probabilities assigned for each word in the vocabulary. It will determine what is the probability that word A is assigned to topic 1, 2, 3. Based on this probability distribution it will analyze the input document to see how much each topic is covered through the document. This part is abstracted from the code by the gensim library we used. Our model is actually saved in the models folder of our project. The model is called from the app.py application and it is generated by the lda jupyter notebook.

The model training is done through the jupyter notebook lda.ipynb file. This file takes in a text file and first does some data cleaning. Specifically it removes spaces punctuation, filler words, stop words. Additionally we ensure the word falls into the category of a noun, verb or adjective to ensure this is a significant word in the transcript. Next we build a dictionary of the words by using a bag of words representation and then assigning weights to each word using a TF-IDF model. Next we set the weights of the model. These weights can be adjusted by the programmer based on experimentation and their needs. The seed variable sets the seed for the random number generator in gibbs sampling performed by lda model. NUM_topics is the number of topics the programmer wants to assign. The alpha sets the document-topic density as alpha goes up, documents contain more topics and visa versa. The eta controls the topic word density, the higher the beta the topics are made up of more words. These can also be thought of as “smoothing” parameters the higher the variables the smoother the distribution. The programmer can tune these variables to best fit their needs and data set. Finally we call a gensim library to train the model. The output is printed to the terminal. This is not seen by the end user but used by the programmer to adjust he model before updating the app.py application.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

To run the code the end user simply needs to
1) Follow this link: https://share.streamlit.io/preetiamin/courseproject/main/app.py
2) Input desired text into the website.

In the actual code we use the following libraries: Panda, Numpy, Gensim, NLTK, Spacey, Os, Streamlit

For training data we used the ami transcript data set: https://groups.inf.ed.ac.uk/ami/corpus/. Additionally, we generated our own transcripts from our own meetings and recordings to provide more data to train with.

If a programmer wanted to update the model with their own transcripts they would need to upload the transcripts to the “ami-transcripts” folder and then run the lda.ipynb program.

4) Brief description of contribution of each team member in case of a multi-person team.

Preeti Amin: Model training and experimentation, build user interface and applications
Sunnie Wang: Model training and experimentation, transcription generation, demo
Kristopher Gallagher: Model training and experimentation, transcription generation, documentation