Data Preprocessing:

The first part of the code is dedicated to preprocessing the data. It starts with a block of text which is cleaned and converted into a list of sequences where each sequence is a string of fixed length. The 'text\_cleaner' function is used to clean the data by converting all text to lowercase, removing possessive pronouns, and replacing any non-alphabet characters with a space. It also removes any words that are less than 3 characters in length. The 'create\_seq' function is then used to convert this cleaned text into sequences of a fixed length.

Character Mapping:

The next part of the code creates a mapping from each unique character in the dataset to a unique integer, and uses this mapping to transform the sequences of characters into sequences of integers.

Train-Test Split:

After encoding the sequences, the last character of each sequence is separated to serve as the label (y), and the remaining part of each sequence is used as the input data (X). These are then split into a training set and a validation set.

Model Definition:

The model used here is a type of recurrent neural network (RNN) called a Gated Recurrent Unit (GRU). This type of network is particularly suited to sequence prediction problems. The network uses an embedding layer to convert the integer-encoded characters into dense vectors of fixed size, a GRU layer with 150 units, and a final dense layer that produces a probability prediction for each character in the vocabulary.

Model Training:

The model is compiled and then trained using the training data. The Adam optimizer and categorical cross-entropy loss function are used. The model's accuracy on the validation data is monitored during training, and training will stop if the validation loss does not decrease after a certain number of epochs (controlled by the 'EarlyStopping' callback). The 'ModelCheckpoint' callback is used to save the weights of the model at the end of each epoch if the model's performance on the validation data has improved.

Text Generation:

The final part of the code is a function 'generate\_seq' which generates new text given a seed text. The function works by encoding the seed text, then using the model to predict the next character, and updating the seed text with this character. This process is repeated a specified number of times. The function returns the top 3 predictions at each step, showing the softmax score, the integer token, the character string, and the resulting sequence for each prediction.