**Function Documentation**

1. **import nltk:** This line imports the Natural Language Toolkit (NLTK) library, which provides tools for working with human language data.
2. **import numpy as np:** Here, we import the NumPy library and alias it as np. NumPy is commonly used for numerical computations in Python.
3. **import random:** This line imports the built-in random module, which allows you to generate random numbers and make random selections.
4. **import string:** The string module provides various string-related functions and constants.
5. **import bs4 as bs:** This imports the Beautiful Soup library (usually aliased as bs) for web scraping and parsing HTML/XML documents.
6. **import urllib.request:** The urllib.request module is used for opening URLs and fetching data from the web.
7. **import re:** The re module provides regular expression operations for pattern matching and manipulation.
8. **from nltk.corpus import stopwords:** NLTK provides a list of common stopwords (e.g., “the,” “and,” “is”) that are often removed from text during natural language processing tasks.
9. **from nltk.tokenize import word\_tokenize:** This imports the word\_tokenize function from NLTK, which splits text into individual words (tokens).
10. **from nltk.stem import WordNetLemmatizer:** The WordNetLemmatizer is used for lemmatization, which reduces words to their base or dictionary form (e.g., “running” to “run”).

Let’s break down the code snippet you provided:

1. **get\_wikipedia\_data(url):**
   * This function fetches data from a given Wikipedia URL and returns the parsed text.
   * It does the following:
     + Opens the URL using urllib.request.urlopen(url).
     + Reads the content of the URL using .read().
     + Parses the content using Beautiful Soup (bs4) with the 'lxml' parser.
     + Extracts all paragraphs (<p>) from the parsed content.
     + Concatenates the text from all paragraphs into a single string.
     + Converts the text to lowercase.
     + Removes any square bracket notation (e.g., [1], [2]) using regular expressions.
   * The resulting cleaned text is returned.
2. **preprocess\_text(text):**
   * This function tokenizes, removes stop words and punctuation, and lemmatizes the input text.
   * It performs the following steps:
     + Tokenizes the input text using word\_tokenize.
     + Retrieves a set of English stopwords using stopwords.words('english').
     + Converts words to lowercase and filters out non-alphanumeric tokens.
     + Lemmatizes each word using the WordNet lemmatizer.
     + Returns the cleaned list of words.
3. **train\_chatbot\_model(corpus):**
   * This function trains a chatbot model using TF-IDF vectorization and a neural network.
   * It does the following:
     + Creates a TfidfVectorizer to convert the corpus (list of text documents) into a matrix of TF-IDF features.
     + Fits the vectorizer to the corpus and transforms it into a sparse matrix X.
     + Creates dummy target values y assuming that all questions have unique answers (one-hot encoding).
     + Defines a neural network model using Keras:
       - Input layer with 512 neurons and ReLU activation.
       - Dropout layer with a dropout rate of 0.5.
       - Hidden layer with 256 neurons and ReLU activation.
       - Another dropout layer.
       - Output layer with as many neurons as there are documents in the corpus, using softmax activation.
     + Compiles the model with Adam optimizer and categorical cross-entropy loss.
     + Fits the model to the training data (X) and target values (y) for 1000 epochs with a batch size of 32.
     + Returns the trained model and the vectorizer.
4. **predict\_answer(question, model, vectorizer, corpus):**
   * This function predicts an answer to a given question using a trained model.
   * Here’s how it works:
     + It takes four parameters:
       - question: The user’s input question.
       - model: The trained chatbot model (presumably trained on the Wikipedia data).
       - vectorizer: A TF-IDF vectorizer used to convert text into numerical features.
       - corpus: A list of sentences (or text documents) used during training.
     + It performs the following steps:
       - Transforms the input question into a vector using the vectorizer.
       - Predicts the answer probabilities using the trained model.
       - Finds the index of the most probable answer using np.argmax(predictions).
       - Retrieves the corresponding answer from the corpus.
       - Returns the predicted answer.
5. **chat():**
   * This is the main function that allows interaction with the chatbot.
   * It does the following:
     + Sets the Wikipedia URL for the Mercedes-Benz page.
     + Fetches and preprocesses the Wikipedia data (similar to what we discussed earlier).
     + Creates a corpus by tokenizing the data into sentences.
     + Trains the chatbot model using the train\_chatbot\_model function.
     + Enters a loop where the user can input questions:
       - If the user types “exit,” the loop breaks.
       - Otherwise, the chatbot predicts an answer based on the user’s question and prints it.

Overall, this simple chatbot that interacts with users based on information from the Mercedes-Benz Wikipedia page. The chatbot uses TF-IDF vectorization and a neural network for prediction.

**Note:-** To improve the accuracy of chatbot after additional training, we can focus on fine-tuning the model and optimizing its performance .