The code provided is for training a chatbot using a neural network. Let's go through it step by step:

1. Importing necessary libraries:

```
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random
```

The code begins by importing the required libraries, including NLTK (Natural Language Toolkit) for natural language processing, Keras for building the neural network model, and other necessary libraries.

2. Data preprocessing:

```
nltk.download('punkt')
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
```

The NLTK **'punkt'** and **'wordnet'** resources are downloaded, which are essential for tokenizing words and lemmatization, respectively. The WordNetLemmatizer is then initialized to perform lemmatization.

3. Loading and preparing the data:

```
words = []
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('data.json', encoding='utf-8').read()
intents = json.loads(data_file)
```

The code initializes empty lists for words, classes, and documents. It defines a list of characters to ignore, such as question marks and exclamation marks. It reads the content of the 'data.json' file, which contains the intents and their corresponding patterns and responses for the chatbot.

4. Preprocessing the data:

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        documents.append((w, intent['tag']))

        if intent['tag'] not in classes:
            classes.append(intent['tag'])

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
classes = sorted(list(set(classes)))
```

The code iterates over each intent and its patterns. It tokenizes the words in each pattern using **'nltk.word_tokenize()'**, adds them to the **'words'** list, and creates a tuple of the words and the intent tag, which is then appended to the **'documents'** list. It also adds the intent tag to the **'classes'** list if it is not already present.

After that, the code performs lemmatization on the words, converts them to lowercase, removes the ignored words, and sorts and converts the **'words'** and **'classes'** lists into unique sets.

5. Saving the processed data:

```
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

The code saves the **'words'** and **'classes'** lists using the pickle module, which serializes the objects and stores them in **'words.pkl'** and **'classes.pkl'** files respectively. These files will be used later during model training and prediction.

6. Creating training data:

```
training = []
output_empty = [0] * len(classes)
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    training.append([bag, output_row])

random.shuffle(training)
training = np.array(training)
train_x = list(training[:, 0])
train_y = list(training[:, 1])
```

The code prepares the training data by creating a bag-of-words representation. It initializes an empty **'training'** list and an **'output_empty'** list with zeros corresponding to the number of classes. For each document, it creates a bag-of-words vector by checking if each word in the **'words'** list exists in the document's pattern. It then creates the corresponding output row with a 1 at the index of the document's class in the **'classes'** list. The bag-of-words vector and output row are appended to the **'training'** list.

The training data is shuffled randomly to avoid any bias, and then converted into NumPy arrays. The **'train_x'** and **'train_y'** lists are created to store the patterns (input) and intents (output) respectively.

7. Creating an training the model:

```
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)
```

The code defines a sequential model using Keras. It consists of multiple layers, including dense (fully connected) layers and dropout layers for regularization. The model is compiled with the stochastic gradient descent (SGD) optimizer and categorical cross-entropy loss function.

The model is then trained using the training data (**'train_x'** and **'train_y'**) for a specified number of epochs and batch size. The training history is stored in **'hist'**, and the trained model is saved as **'chatbot_model.h5'**.

8. Finalizing the training process:

```
print("model created")
```

This line simply indicates that the model training process has been completed.

That's the explanation of the provided code. It preprocesses the data, prepares the training data, creates and trains a neural network model using Keras, and saves the trained model.