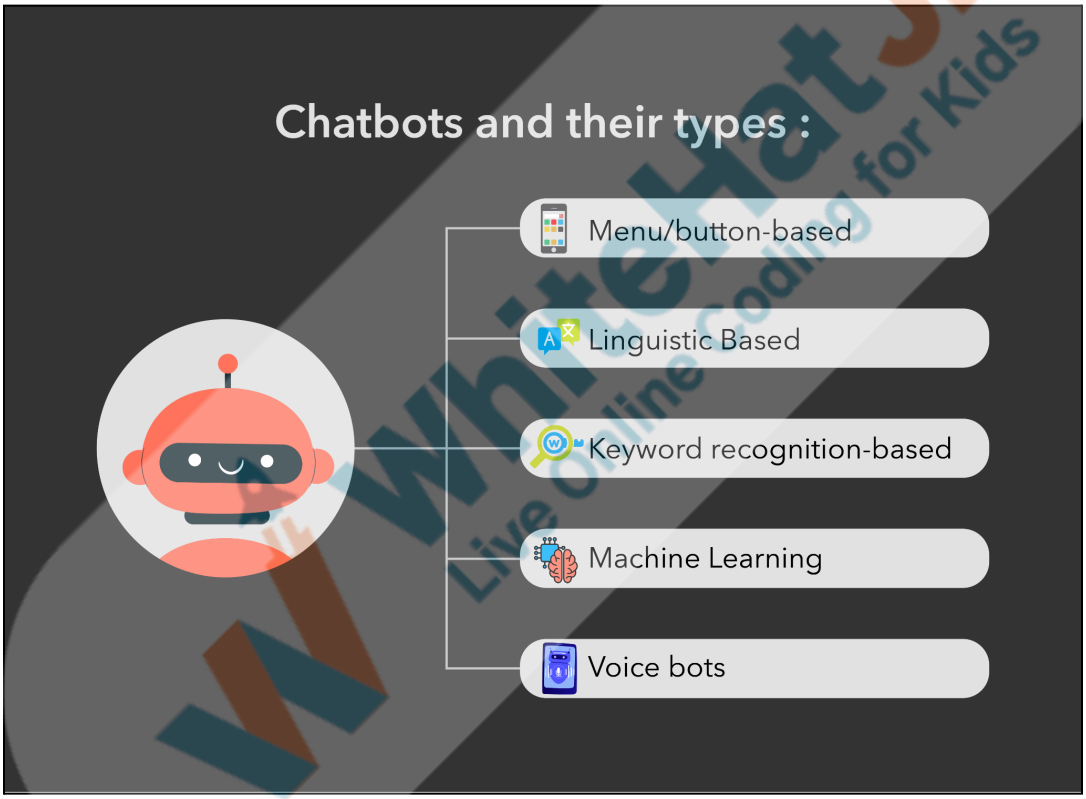
Topic	TESTING CHATBOT	
Class Description	**The student will learn to create a model for training and testing the chatbot.**	
Class	**PRO C120**	
Class time	**45 mins**	
Goal	• Define a model for chatbot. • Train the model. • Test Chatbot responses.	
Resources Required	• Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone • Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit & Permissions:	NLTK by Team NLTK	
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow **Slide 1 to 4** Refer to speaker notes and follow the instructions on each slide.		

Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Can you tell me what we learned in the previous class? ***Note**: Encourage the student to give answers and be more involved in the discussion.* Amazing! We created a json file named 'intents.json'. Also, we performed stemming on words to find their root words. We had created a bag of words and a training dataset. Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! **ESR**: we created a training dataset for the chatbot. Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 5 to 10** 	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Can you give me an example of a chatbot? ***Note:** Depending upon the student's answer categorize the*	**ESR:** Varied

chatbot as given below.

Yes! So there are different types of chatbots available. Some take voice as input and in some of the chatbots, we need to type the message and send it.

Depending upon the type of input from the user and how the responses are processed, the chatbots are classified as follows:



1. **Menu or button-based:**
   In Menu-based chatbots, answers/responses are provided in the form of a menu.
2. **Linguistic Based:**
   Linguistic-based uses if-then logic to resolve customer queries.
3. **Keyword recognition based:**
   It takes the user input and uses keywords and NLP

<table>
<tr><td>determines the response.
4. Machine Learning:
Machine Learning chatbots take the previous conversation as data for learning and depending upon the context give the response.
5. Voice bots:
Voice bots are voice-activated chatbots. Here the user input is taken in the form of voice and accordingly response is given. Amazon’s Alexa is an example of a voice bot.

Can you tell me under which category does our chatbot falls?

Yes!! We’ll be creating a Machine Learning model which will be trained on the dataset we provide in intents.json.

Let’s continue with creating the model.</td><td>ESR: Machine Learning-based</td></tr>
<tr><td colspan="2">Teacher Ends Slideshow</td></tr>
<tr><td colspan="2">TEACHER-LED ACTIVITY - 10 mins</td></tr>
<tr><td colspan="2">Teacher Initiates Screen Share</td></tr>
<tr><td colspan="2"><u>ACTIVITY</u>
• Create a CNN model to train a chatbot.</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>Let’s start by creating the model file for training the model.

Note: Open Teacher Activity1 for boilerplate code. Show the file structure to the student.

Intents.json is prepared with the tags, patterns and responses.</td><td></td></tr>
</table>

In the file **data_preprocessing** we have the pre-processed data **train_x** and **train_y** as we had processed in the previous class.

Here we'll create two files one **train_bot** for defining the model and another **predict_response** to check the responses of the chatbot.

intents.json > [ ] intents > {} 4 > [ ] responses > abc 1

```
{"intents": [
    {"tag": "greeting",
     "patterns": ["Hi there!", "How are you?", "Is anyone there?",
                  "Hey!","Hola!", "Hello!"],
     "responses": ["Hello, thanks for asking!", "Good to see you.",
                   "Hi there, how can I help?"]
    },
    {"tag": "goodbye",
     "patterns": ["Bye.", "See you later!", "Goodbye!",
                  "Nice chatting to you, bye.", "Till next time!!"],
     "responses": ["See you!", "Have a nice day.", "Bye! Come back again soon."]
    },
    {"tag": "thanks",
     "patterns": ["Thanks!", "Thank you", "That's helpful",
                  "Awesome, thanks", "Thanks for helping me"],
     "responses": ["Happy to help!", "Any time!", "My pleasure."]
    },
    {"tag": "noanswer",
     "patterns": [""],
     "responses": ["Sorry, can't understand you", "Please give me more info",
                   "Not sure I understand"]
    },
    {"tag": "options",
     "patterns": ["I am bored today!", "What you can do?",
                  "I don't know what to do", "What help you provide?",
                  "How you can be helpful?", "What support is offered"],
     "responses": ["I can guide you through an activity to spend the day. What's your hobby?",
                   "we can plan out for a daily activity. what would you like doing?"]
    },
    {"tag": "art",
        "patterns": ["I like art", "I like craft", "art and craft", "drawing", "sketching?", "painting"],
        "responses": ["you can draw a geometric art and color it!!","you can sketch a scenery today.",
                      "Try Dotted Mandala art today!"]
    },
    {"tag": "music",
        "patterns": ["I like music", "I like songs", "songs and music", "pop"],
        "responses": ["you can listen to songs on FM radio",
                      "You can record a video to post on social media website"]
    },

```

```
    {"tag": "cooking",
            "patterns": ["I like cooking", "I like baking", "eating pizza", "cooking","food"],
            "responses": ["you can try baking a cake today!","You can cook a pizza today"]},

    {"tag": "fitness",
        "patterns": ["I like to work out", "I like aerobics", "exercise"],
        "responses": ["you can try zumba dance today !","You can try pilates today"]},

    {"tag": "travel",
        "patterns": ["I like to travel", "I like exploring different places", "travelling"],
        "responses": ["you can visit a zoo today!","You can visit an amusement park today"]}
]}
```

The next file is **data_preprocessing.** This file has the code of the previous class where the preprocessed training data was created using stemming and Bag Of Words (BOW).

data_preprocessing.py > get_stem_words

```
# Text Data Preprocessing Lib
import nltk

from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
import json
import pickle
import numpy as np

words=[] #list of unique roots words in the data
classes = [] #list of unique tags in the data
#list of the pair of (['words', 'of', 'the', 'sentence'], 'tags')
pattern_word_tags_list = []
ignore_words = ['?', '!',',','.', "'s", "'m"]

train_data_file = open('intents.json').read()
intents = json.loads(train_data_file)

```

```
train_data_file = open('intents.json').read()
intents = json.loads(train_data_file)

def get_stem_words(words, ignore_words):
    stem_words = []
    for word in words:
        if word not in ignore_words:
            w = stemmer.stem(word.lower())
            stem_words.append(w)
    return stem_words
```

Note:- The following function is the combination of the code to get the pattern_word_tags_list , stem_words and classes seen in last class.

```
def create_bot_corpus(words, classes, pattern_word_tags_list, ignore_words):

    for intent in intents['intents']:

        # Add all patterns and tags to a list
        for pattern in intent['patterns']:
            pattern_word = nltk.word_tokenize(pattern)
            words.extend(pattern_word)
            pattern_word_tags_list.append((pattern_word, intent['tag']))

        # Add all tags to the classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

    stem_words = get_stem_words(words, ignore_words)
    stem_words = sorted(list(set(stem_words)))
    classes = sorted(list(set(classes)))

    return stem_words, classes, pattern_word_tags_list
```

```
def bag_of_words_encoding(stem_words, pattern_word_tags_list):
    bag = []
    for word_tags in pattern_word_tags_list:

        pattern_words = word_tags[0]
        bag_of_words = []
        stem_pattern_words= get_stem_words(pattern_words, ignore_words)
        for word in stem_words:
            if word in stem_pattern_words:
                bag_of_words.append(1)
            else:
                bag_of_words.append(0)
        bag.append(bag_of_words)
    return np.array(bag)

```

```python
def class_label_encoding(classes, pattern_word_tags_list):
    labels = []
    for word_tags in pattern_word_tags_list:

        labels_encoding = list([0]*len(classes))
        tag = word_tags[1]
        tag_index = classes.index(tag)
        labels_encoding[tag_index] = 1
        labels.append(labels_encoding)
    return np.array(labels)
```

```python
def preprocess_train_data():

    stem_words, tag_classes, word_tags_list = create_bot_corpus(words, classes,
                                              pattern_word_tags_list, ignore_words)
    pickle.dump(stem_words, open('words.pkl','wb'))
    pickle.dump(tag_classes, open('classes.pkl','wb'))
    train_x = bag_of_words_encoding(stem_words, word_tags_list)
    train_y = class_label_encoding(tag_classes, word_tags_list)

    return train_x, train_y
```

1. Go to the command prompt and traverse the working folder as we did in class 110.
2. Create a virtual environment **(Windows/Mac)** using **python -m venv <name_of_the _environment>**for testing the model.
3. Activate the virtual environment using the following command:

```
<name_of_the_environment>\Scripts\activate
```

Note: *Run the command to create a virtual environment with the name "**AI_chatbot**". The environment is user-defined. We can keep the name as we want relevant to our project.*

```
C:\Whitehat_jr\PRO-C120>python -m venv Test_chatbot

C:\Whitehat_jr\PRO-C120>Test_chatbot\Scripts\activate
```

4. Install NLTK using the command **pip install nltk.** Also, install all the required libraries in the environment such as **numpy, tensorflow** etc.

```
(Test_chatbot) C:\Whitehat_jr\PRO-C120>pip install nltk
```

5. Run the python file data preprocessing to get the preprocessed training dataset prepared for the model.

```
(Test_chatbot) C:\Whitehat_jr\PRO-C120>python data_preprocessing.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Now next step is to create a file to train the model. We'll name this file as **train_bot.py.**

In this file we'll create the model and train it on the dataset prepared.

1. Import sequential from **ternsorflow.keras.models** to create CNN model as we did before.
2. Import **Dense, Activation** and **Dropout** layers. Since the model will be trained on a small dataset, we'll need less number of layers than we used before.
3. Import **Adam** optimizer.
   Optimizers are used to reduce losses while training the model.

```python
# Model Training Lib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam

from data_preprocessing import preprocess_train_data
```

Create a function for defining the CNN model. This function **train_bot_model()** takes **train_x** and **train_y** as parameters.

1. Define the model being sequential using the **Sequential()** method.
2. The very first layer we'll add is a **Dense layer** with **128** output units. Input will be the training data that is **train_x** and activation function as 'relu'.
3. Add Dropout layer with **0.5** dropout.
4. The second layer is also a **Dense layer** with **64** output units. Activation function as 'relu'.
5. Add **Dropout layer** with **0.5** dropout.
6. The last layer will be the dense layer with output units equal to the number of tags. Use **softmax** activation function as its last layer of our model.

```python
# Model Training Lib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam

from data_preprocessing import preprocess_train_data

def train_bot_model(train_x, train_y):
    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'))

```

After defining the model, what is the next step that we follow?
Yes!! So we'll follow these steps in the similar manner as we have done before.

7. **Compile** the model by defining **loss, optimizer** and **metrics.**
8. The next step would be to **fit and save** the model. Provide all the necessary parameters to fit the model. (**training data, no of epochs, batch size and verbose**)
9. Save the model by name **chatbot_model.h5.**
10. After saving the model file print a message '**Model File Created & Saved'.**

ESR: Compile, fit and save the model

train_bot.py > ...

```python
# Model Training Lib
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam

from data_preprocessing import preprocess_train_data

def train_bot_model(train_x, train_y):
    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'))

    # Compile Model
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])

    # Fit & Save Model
    history = model.fit(train_x, train_y, epochs=200, batch_size=5, verbose=True)
    model.save('chatbot_model.h5', history)
    print("Model File Created & Saved")
```

Now run this file for creating and saving the model.

```
(Test_chatbot) C:\Whitehat_jr\PRO-C120>python train_bot.py
2022-01-27 17:53:43.243961: W tensorflow/stream_executor/platform/default/dso_loader.
 found
2022-01-27 17:53:43.244671: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignc
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
2022-01-27 17:53:47.212220: W tensorflow/stream_executor/platform/default/dso_loader.
2022-01-27 17:53:47.212283: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] fai
2022-01-27 17:53:47.218321: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169
2022-01-27 17:53:47.218730: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176
2022-01-27 17:53:47.221900: I tensorflow/core/platform/cpu_feature_guard.cc:151] This
 the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
Epoch 1/200
9/9 [==============================] - 0s 1ms/step - loss: 2.3440 - accuracy: 0.0909
Epoch 2/200
9/9 [==============================] - 0s 0s/step - loss: 2.3171 - accuracy: 0.0909
Epoch 3/200
9/9 [==============================] - 0s 2ms/step - loss: 2.2869 - accuracy: 0.1591
Epoch 4/200
9/9 [==============================] - 0s 0s/step - loss: 2.1843 - accuracy: 0.3409
```

Thus, we have successfully created the model for prediction. You can see the model file is created which will be used for the prediction of labels (tags).	

Teacher Stops Screen Share

Please share your screen with me.	

Teacher Starts Slideshow

Slide 11 to 14

Refer to speaker notes and follow the instructions on each slide.

STUDENT-LED ACTIVITY - 20 mins

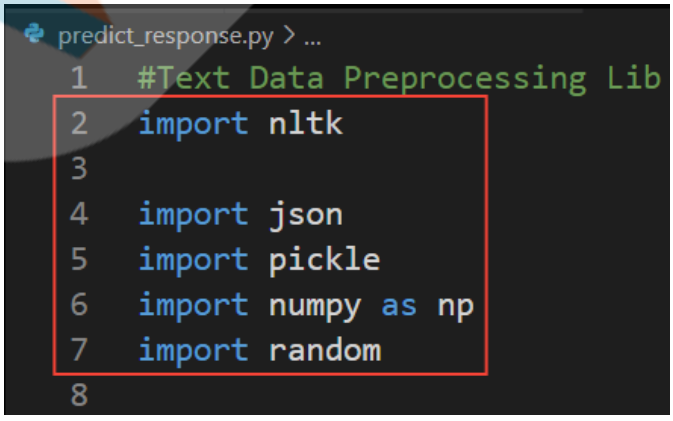
- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

ACTIVITY

- **Preprocess text data entered by user**
- **Create a function for chatbot prediction**

Teacher Action	Student Action
Open *Student activity 1* to check the boilerplate code. The **train_bot** file is given to you. Download all the files in a folder and run the file in the following order. 1. Create a virtual environment and install all the necessary libraries such as **nltk, tensorflow and numpy.** ***Note:*** *Student can add more responses to intents.json file. This should be done before preprocessing the data.* 2. Now run the **data_preprocessing** file for creating the training dataset.**(train_x and train_y).** 3. Run **train_bot** file for creating and saving the model file. Next, create a file called **predict_response** to take user input and give a response through the chatbot. 1. Import **nltk**. From nltk download **punkt** and **wordnet**. 2. Download **json**, **numpy**, **random** and **pickle**. These files are necessary for processing data.	



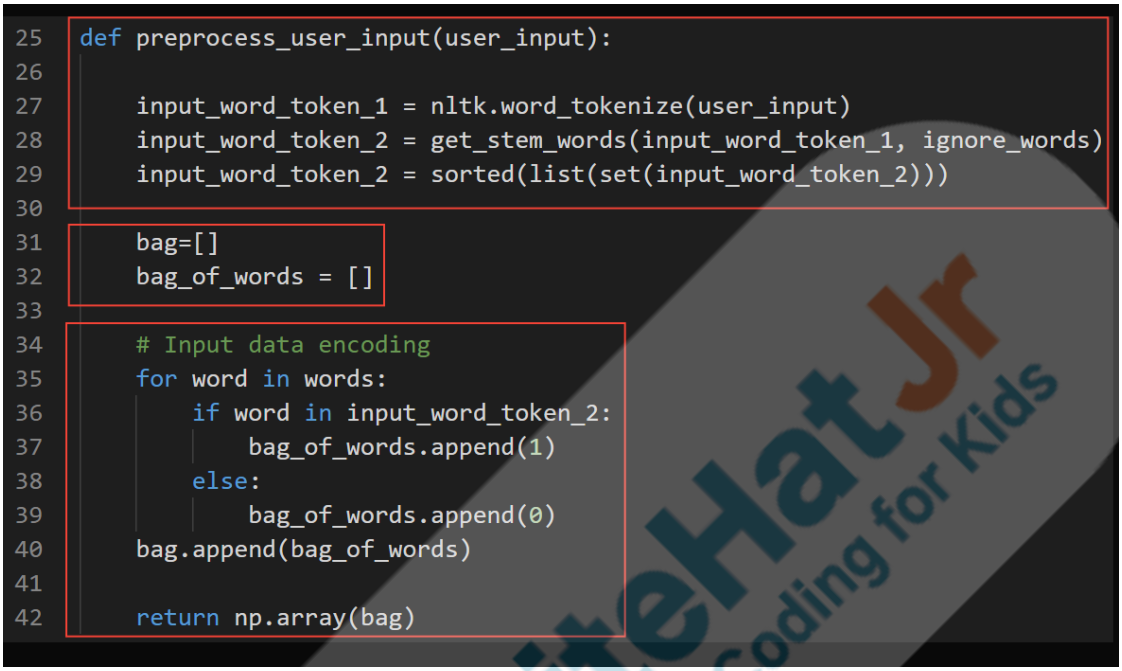
3. Create a list of **ignore_words**.
4. Import **tensorflow.**
5. Now the user input should also be converted into an array of stem words. Import **get_stem_words** from **data_preprocessing**.
6. We will load the model file.
7. Load all the data files. The **intents.json** has the raw dataset.
8. The **classes.pkl** and **words.pkl** files have the preprocessed training dataset.

predict_response.py > ...

```
#Text Data Preprocessing Lib
import nltk

import json
import pickle
import numpy as np
import random

ignore_words = ['?', '!',',','.', "'s", "'m"]

# Model Load Lib
import tensorflow
from data_preprocessing import get_stem_words

model = tensorflow.keras.models.load_model('./chatbot_model.h5')

# Load data files
intents = json.loads(open('./intents.json').read())
words = pickle.load(open('./words.pkl','rb'))
classes = pickle.load(open('./classes.pkl','rb'))

```

Now write a function to preprocess the text input we get from the user:

1. Create a function called **preprocess_user_input.** This function takes user input, tokenizes it, converts the tokenized text into stem words.
2. The sorted list of stem words is stored in **input_word_token_2.**
3. Now, the **Bag Of words** is created using this list. If the word in the given sentence is present in the list of stem words then '**1**' is appended in **bag_of_words** otherwise '**0**' is appended.
4. This **bag_of_words** is then converted into a **numpy**

array.	
 ``` def preprocess_user_input(user_input): input_word_token_1 = nltk.word_tokenize(user_input) input_word_token_2 = get_stem_words(input_word_token_1, ignore_words) input_word_token_2 = sorted(list(set(input_word_token_2))) bag=[] bag_of_words = [] # Input data encoding for word in words: if word in input_word_token_2: bag_of_words.append(1) else: bag_of_words.append(0) bag.append(bag_of_words) return np.array(bag) ```	
Create another function for the prediction of class label. This label will be the predicted tag. So, the response will be chosen from the predicted tag: 1. Create a **bot_class_prediction** function. This function takes user input as its parameters. 2. Inside this function, the **preprocess_user_input** function is being called that gives the preprocessed text for **user_input.** 3. Use the **predict** function to predict the label and store it in the **prediction** variable. 4. This prediction variable has an array of predicted classes with their probabilities. The maximum value is found by using the **argmax()** function. 5. So, this function returns the prediction as to the class label or tag.	

```
44  def bot_class_prediction(user_input):
45
46      inp = preprocess_user_input(user_input)
47      prediction = model.predict(inp)
48      predicted_class_label = np.argmax(prediction[0])
49      return predicted_class_label
50
```

At last, create a function to give a response according to the user input.

1. In this function call the function for prediction to get the **predicted_class_label.**
2. **predicted _class** is the **predicted tag**. Now to get the predicted tag we'll use the **classes.pkl** file in which we had stored these tags with respective labels.
3. Next, loop through the **intents** in the intents.json file to compare the predicted tag with all the tags.
4. **bot_response** is chosen randomly amongst the available responses under the predicted tag.
5. Thus, whenever the user gives an input, it is being checked under which tag it falls and accordingly the response is given from bot.

```
52  def bot_response(user_input):
53
54      predicted_class_label =  bot_class_prediction(user_input)
55      predicted_class = classes[predicted_class_label]
56
57      for intent in intents['intents']:
58       if intent['tag']==predicted_class:
59           bot_response = random.choice(intent['responses'])
60           return bot_response
61
```

Let's check the response from the chatbot now.

1. Print the sentence **'Hi, I am Stella, How can I help you?.**
2. Use a **while loop** to check the user input continuously.
3. Store user input in the **user_input** variable.
4. Pass this variable to the function **bot_response.**

5. **response** variable stores the chatbot response. Print this response.
6. Since we have used a **while** loop, the process will continue.

```
62  print("Hi I am Stella, How Can I help you?")
63
64  while True:
65      user_input = input("Type your message here:")
66      print("User Input: ", user_input)
67
68      response = bot_response(user_input)
69      print("Bot Response: ", response)
70
```

Let's save and run this file to check how the chatbot works.

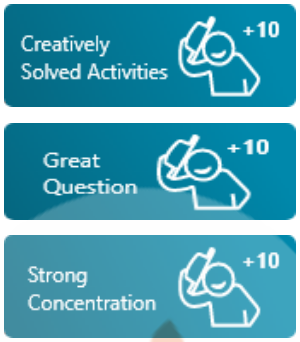
```
Hi I am Stella, How Can I help you?
Type your message here:hi
User Input:  hi
Bot Response:  Hello, thanks for asking!
Type your message here:what do you do?
User Input:  what do you do?
Bot Response:  we can plan out for a daily activity. what would you like doing?
Type your message here:i am bored
User Input:  i am bored
Bot Response:  I can guide you through an activity to spend the day. What's your hobby?
Type your message here:ok. i like drawing
User Input:  ok. i like drawing
Bot Response:  Try Dotted Mandala art today!
Type your message here:thank you
User Input:  thank you
Bot Response:  My pleasure.
Type your message here:what else?
User Input:  what else?
Bot Response:  we can plan out for a daily activity. what would you like doing?
```

Great work!!

So we had created the chatbot successfully and as you can see it gives responses according to the intents.json file.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins	
Teacher Starts Slideshow **Slide 15 to 20** 	
Activity details **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Continue WRAP-UP Session **Slide 21 to 26** 	
Activity Details **Following are the session deliverables:** ● Explain the facts and trivia ● Next class challenge ● Project for the day ● Additional Activity (Optional)	
FEEDBACK ● **Appreciate and compliment the student for trying to learn a difficult concept.** ● **Get to know how they are feeling after the session.** ● **Review and check their understanding.**	
Teacher Action	**Student Action**
You get “hats-off” for your excellent work!	*Make sure you have given at least 2 hats-off during the class for:*

In the next class, we'll be creating a capstone project using all the concepts we have learned before.	
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks 	

Activity Name	Description	Links
Teacher Activity 1	Boilerplate Code	https://github.com/procodingclass/PR

		O-C120-Teacher-Boilerplate-Code
Teacher Activity 2	Reference Code	https://github.com/procodingclass/PRO-C120-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/85a3cb65-6734-488c-a69f-3e34491ac955.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C120-Project-Solution.git
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/47400df7-110f-408e-8881-628a12c85f4f.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/fe4266bc-90b9-4ab8-9613-f23e7940683a.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C120-Student-Boilerplate-Code