Phase 5: Project Documentation & Submission

Problem statement:

Create a chatbot in Python

Problem Definition:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

Design Thinking:

Functionality: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.

User Interface: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.

Natural Language Processing (NLP): Implement NLP techniques to understand and process user input in a conversational manner.

Responses: Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.

Integration: Decide how the chatbot will be integrated with the website or app.

Testing and Improvement: Continuously test and refine the chatbot's performance based on user interactions.

Phases of development:

Phase 1:

Understanding the problem statement and creating a document how to solve the problem. And think on design and present in form of a document.

Phase 2:

In this phase, we can explore innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

Consider exploring advanced techniques like pre-trained language models (e.g., GPT-3) to enhance the quality of response.

Phase 3:

In this phase, begin building your project by loading and pre-processing the dataset the dataset.

Starting building the chatbot by preparing the environment and implementing basic user interactions. Install required libraries, like transformers for GPT-3 integration and flask for web app development.

Phase 4:

In this phase, continue building the chatbot by integrating it into a web app using Flask.

Phase 5:

In this phase, document the project and prepare it for submission.

Libraries:

1.NumPy and Pandas

NumPy is a library for Python that adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Pandas is a high-level data manipulation tool that is built on the NumPy package.

2.TensorFlow

TensorFlow allows you to create dataflow graphs that describe how data moves through a graph. The graph consists of nodes that represent a mathematical operation. A connection or edge between nodes is a multidimensional data array.

3.Matplotlib and Seaborn

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

4.Flask

Flask is a lightweight Python web framework that provides useful tools and features for creating web applications in the Python Language. It gives developers flexibility and is an accessible framework for new developers because you can build a web application quickly using only a single Python file.

Chatbot interaction:

They are designed to simulate human conversation, allowing users to ask questions, seek assistance, or complete tasks seamlessly. The significance of chatbot communication lies in their ability to engage users in a personalized and interactive manner while providing valuable support and guidance.

Dataset Source: kaggle

https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

Source Code:

Importing libraries

```
#model
import tensorflow as tf
from sklearn.model_selection import train_test_split

#nlp processing
import unicodedata
import re
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

Data pre-processing

The basic text processing in NLP are:

1. Sentence Segmentation
2. Normalization
3. Tokenization

1.Segmentation

```
#reading data
data=open('D:\\dialogs.txt','r').read()
```

```
#paried list of question and corresponding answer
QA_list=[QA.split('\t') for QA in data.split('\n')]
print(QA_list[:5])
```

Output:

[['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yourself?", "i'm pretty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no problem. so how have you been?'], ['no problem. so how have you been?', "i've been great. what about you?"], ["i've been great. what about you?", "i've been good. i'm in school right now."]]

```
questions=[row[0] for row in QA_list]
answers=[row[1] for row in QA_list]
print(questions[0:5])
print(answers[0:5])
```

Output:

['hi, how are you doing?', "i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how have you been?', "i've been great. what about you?"]
["i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how have you been?', "i've been great. what about you?", "i've been good. i'm in school right now."]

2.Normalization

```
def remove_diacritic(text):
    return ''.join(char for char in unicodedata.normalize('NFD',text)
                   if unicodedata.category(char) !='Mn')
```

```
def preprocessing(text):

    #Case folding and removing extra whitespaces
    text=remove_diacritic(text.lower().strip())

    #Ensuring punctuation marks to be treated as tokens
    text=re.sub(r"([?.!,¿])", r" \1 ", text)

    #Removing redundant spaces
    text= re.sub(r'[" "]+', " ", text)

    #Removing non alphabetic characters
    text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)

    text=text.strip()

    #Indicating the start and end of each sentence
    text='<start> ' + text + ' <end>'

    return text
```

```
preprocessed_questions=[preprocessing(sen) for sen in questions]
preprocessed_answers=[preprocessing(sen) for sen in answers]

print(preprocessed_questions[0])
print(preprocessed_answers[0])
```

Output:

<start> hi , how are you doing ? <end>
<start> i m fine . how about yourself ? <end>

3.Tokenization

```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')

    #build vocabulary on unique words
    lang_tokenizer.fit_on_texts(lang)

    return lang_tokenizer
```

Word Embedding

```
def vectorization(lang_tokenizer,lang):

    #word embedding for training the neural network
    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                           padding='post')

    return tensor
```

Creating Dataset

```
def load_Dataset(data,size=None):

    if(size!=None):
        y,X=data[:size]
    else:
        y,X=data

    X_tokenizer=tokenize(X)
    y_tokenizer=tokenize(y)

    X_tensor=vectorization(X_tokenizer,X)
    y_tensor=vectorization(y_tokenizer,y)

    return  X_tensor,X_tokenizer, y_tensor, y_tokenizer
```

```
size=30000
data=preprocessed_answers,preprocessed_questions\

X_tensor,X_tokenizer, y_tensor, y_tokenizer=load_Dataset(data,size)
# Calculate max_length of the target tensors
max_length_y, max_length_X = y_tensor.shape[1], X_tensor.shape[1]
```

Splitting Data

```
X_train, X_val, y_train, y_val = train_test_split(X_tensor, y_tensor, test_size=0.2)

# Show length
print(len(X_train), len(y_train), len(X_val), len(y_val))
```

Output:

```
2980 2980 745 745
```

Tensorflow Dataset

```
BUFFER_SIZE = len(X_train)
BATCH_SIZE = 64
steps_per_epoch = len(X_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(X_tokenizer.word_index)+1
vocab_tar_size = len(y_tokenizer.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((X_train,
y_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Output:

```
(TensorShape([64, 24]), TensorShape([64, 24]))
```

Buliding Model

Encoder

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
```

```
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units)
{}'.format(sample_output.shape))
print ('Encoder Hidden state shape: (batch size, units)
{}'.format(sample_hidden.shape))
```

Output:

```
Encoder output shape: (batch size, sequence length, units) (64, 24, 1024)
Encoder Hidden state shape: (batch size, units) (64, 1024)
```

Attention Mechanism

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis to calculate the
score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length,
units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
```

```
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output)

print("Attention result shape: (batch size, units) {}".format(attention_result.shape))
print("Attention weights shape: (batch_size, sequence_length, 1)
{}".format(attention_weights.shape))
```

Output:

```
Attention result shape: (batch size, units) (64, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (64, 24, 1)
```

Decoder

```
class Decoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc = tf.keras.layers.Dense(vocab_size)

    # used for attention
    self.attention = BahdanauAttention(self.dec_units)

  def call(self, x, hidden, enc_output):
    # enc_output shape == (batch_size, max_length, hidden_size)
    context_vector, attention_weights = self.attention(hidden, enc_output)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)

    # x shape after concatenation == (batch_size, 1, embedding_dim +
hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))
```

```
        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                      sample_hidden, sample_output)

print ('Decoder output shape: (batch_size, vocab size)
{}'.format(sample_decoder_output.shape))
```

Output:

Decoder output shape: (batch_size, vocab size) (64, 2349)

Training Model

1. Pass the input through the encoder which return encoder output and the encoder hidden state.
2. The encoder output, encoder hidden state and the decoder input (which is the start token) is passed to the decoder.
3. The decoder returns the predictions and the decoder hidden state.
4. The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
5. Use teacher forcing to decide the next input to the decoder.
6. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
7. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

```
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([y_tokenizer.word_index['<start>']] * BATCH_SIZE,
1)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

            loss += loss_function(targ[:, t], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss
```

```
EPOCHS = 40

for epoch in range(1, EPOCHS + 1):
    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss

    if(epoch % 4 == 0):
        print('Epoch:{:3d} Loss:{:.4f}'.format(epoch,
                                              total_loss / steps_per_epoch))
```

Output:

```
Epoch:  4 Loss:1.5338
Epoch:  8 Loss:1.2803
Epoch: 12 Loss:1.0975
Epoch: 16 Loss:0.9404
Epoch: 20 Loss:0.7773
Epoch: 24 Loss:0.6040
Epoch: 28 Loss:0.4042
Epoch: 32 Loss:0.2233
Epoch: 36 Loss:0.0989
Epoch: 40 Loss:0.0470
```

Model Evaluation

```
def remove_tags(sentence):
    return sentence.split("<start>")[-1].split("<end>")[0]

def evaluate(sentence):
    sentence = preprocessing(sentence)

    inputs = [X_tokenizer.word_index[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                           maxlen=max_length_X,
                                                           padding='post')
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([y_tokenizer.word_index['<start>']], 0)

    for t in range(max_length_y):
        predictions, dec_hidden, attention_weights = decoder(dec_input,
                                                             dec_hidden,
                                                             enc_out)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += y_tokenizer.index_word[predicted_id] + ' '

        if y_tokenizer.index_word[predicted_id] == '<end>':
            return remove_tags(result), remove_tags(sentence)
```

```
        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return remove_tags(result), remove_tags(sentence)


def ask(sentence):
    result, sentence = evaluate(sentence)

    print('Question: %s' % (sentence))
    print('Predicted answer: {}'.format(result))


for i in range(0, 5):
    ask(questions[1])
```

Output:

```
Question: hi, how are you doing?
Predicted answer: i'm fine. how about yourself?

Question:  i m fine . how about yourself ?
Predicted answer: i m pretty good . thanks for asking .

Question: i'm pretty good. thanks for asking.
Predicted answer: no problem. so how have you been?

Question:  no problem. so how have you been?
Predicted answer: i've been great. what about you?

Question:  i've been great. what about you?
Predicted answer: i've been good. i'm in school right now..
```

Creating web application:
Separately creating .css and .html file for web application implementation.

1.Creating style.css

```
:root {
    --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
    --msger-bg: #fff;
    --border: 2px solid #ddd;
    --left-msg-bg: #ececec;
    --right-msg-bg: #579ffb;
  }

  html {
    box-sizing: border-box;
  }

  *,
  *:before,
  *:after {
    margin: 0;
    padding: 0;
    box-sizing: inherit;
  }

  body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-image: var(--body-bg);
    font-family: Helvetica, sans-serif;
  }

  .msger {
    display: flex;
    flex-flow: column wrap;
    justify-content: space-between;
    width: 100%;
    max-width: 867px;
    margin: 25px 10px;
    height: calc(100% - 50px);
    border: var(--border);
    border-radius: 5px;
    background: var(--msger-bg);
    box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
  }

  .msger-header {
```

```css
  /* display: flex; */
  font-size: medium;
  justify-content: space-between;
  padding: 10px;
  text-align: center;
  border-bottom: var(--border);
  background: #eee;
  color: #666;
}

.msger-chat {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
}
.msger-chat::-webkit-scrollbar {
  width: 6px;
}
.msger-chat::-webkit-scrollbar-track {
  background: #ddd;
}
.msger-chat::-webkit-scrollbar-thumb {
  background: #bdbdbd;
}
.msg {
  display: flex;
  align-items: flex-end;
  margin-bottom: 10px;
}

.msg-img {
  width: 50px;
  height: 50px;
  margin-right: 10px;
  background: #ddd;
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  border-radius: 50%;
}
.msg-bubble {
  max-width: 450px;
  padding: 15px;
  border-radius: 15px;
  background: var(--left-msg-bg);
}
.msg-info {
  display: flex;
```

```
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}
.msg-info-name {
  margin-right: 10px;
  font-weight: bold;
}
.msg-info-time {
  font-size: 0.85em;
}

.left-msg .msg-bubble {
  border-bottom-left-radius: 0;
}

.right-msg {
  flex-direction: row-reverse;
}
.right-msg .msg-bubble {
  background: var(--right-msg-bg);
  color: #fff;
  border-bottom-right-radius: 0;
}
.right-msg .msg-img {
  margin: 0 0 0 10px;
}

.msger-inputarea {
  display: flex;
  padding: 10px;
  border-top: var(--border);
  background: #eee;
}
.msger-inputarea * {
  padding: 10px;
  border: none;
  border-radius: 3px;
  font-size: 1em;
}
.msger-input {
  flex: 1;
  background: #ddd;
}
.msger-send-btn {
  margin-left: 10px;
  background: rgb(0, 196, 65);
  color: #fff;
```

```
    font-weight: bold;
    cursor: pointer;
    transition: background 0.23s;
  }
  .msger-send-btn:hover {
    background: rgb(0, 180, 50);
  }

  .msger-chat {
    background-color: #fcfcfe;
    background-image: url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' width='260' height='260' viewBox='0 0 260
260'%3E%3Cg fill-rule='evenodd'%3E%3Cg fill='%23dddddd' fill-
opacity='0.4'%3E%3Cpath d='M24.37 16c.2.65.39 1.32.54 2H21.17l1.17 2.34.45.9-
.24.11V28a5 5 0 0 0 1-2.23 8.94l-.02.06a8 8 0 0 0 1-7.75 6h-20a8 8 0 0 0 1-7.74-6l-
.02-.06A5 5 0 0 0 1-17.45 28v-6.76l-.79-1.58-.44-.9.9-.44.63-.32H-20a23.01 23.01
0 0 1 44.37-2zm-36.82 2a1 1 0 0 0 0-.44.1l-3.1 1.56.89 1.79 1.31-.66a3 3 0 0 1
2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .9
0l2.21-1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .86.02l2.88-1.27a3 3 0 0 1 2.43
0l2.88 1.27a1 1 0 0 0 .85-.02l3.1-1.55-.89-1.79-1.42.71a3 3 0 0 1-2.56.06l-
2.77-1.23a1 1 0 0 0-.4-.09h-.01a1 1 0 0 0-.4.09l-2.78 1.23a3 3 0 0 1-2.56-
.06l-2.3-1.15a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1L.9 19.22a3 3 0 0 1-2.69
0l-2.2-1.1a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-.44.1l-2.21 1.11a3 3 0 0 1-2.69
0l-2.2-1.1a1 1 0 0 0-.45-.11h-.01zm0-2h-4.9a21.01 21.01 0 0 1 39.61 0h-2.09l-
.06-.13-.26.13h-32.31zm30.35 7.68l1.36-.68h1.3v2h-36v-1.15l.34-.17 1.36-
.68h2.59l1.36.68a3 3 0 0 0 2.69 0l1.36-.68h2.59l1.36.68a3 3 0 0 0 2.69 0L2.26
23h2.59l1.36.68a3 3 0 0 0 2.56.06l1.67-.74h3.23l1.67.74a3 3 0 0 0 2.56-.06zM-
13.82 27l16.37 4.91L18.93 27h-32.75zm-.63 2h.34l16.66 5 16.67-5h.33a3 3 0 1 1
0 6h-34a3 3 0 1 1 0-6zm1.35 8a6 6 0 0 0 0 5.65 4h20a6 6 0 0 0 0 5.66-4H-
13.1z'/%3E%3Cpath id='path6_fill-copy' d='M284.37 16c.2.65.39 1.32.54
2H281.17l1.17 2.34.45.9-.24.11V28a5 5 0 0 0 1-2.23 8.94l-.02.06a8 8 0 0 0 1-7.75
6h-20a8 8 0 0 0 1-7.74-6l-.02-.06a5 5 0 0 0 1-2.24-8.94v-6.76l-.79-1.58-.44-.9.9-
.44.63-.32H240a23.01 23.01 0 0 1 44.37-2zm-36.82 2a1 1 0 0 0 0-.44.1l-3.1
1.56.89 1.79 1.31-.66a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0
1 2.69 0l2.2 1.1a1 1 0 0 0 .9 0l2.21-1.1a3 3 0 0 1 2.69 0l2.2 1.1a1 1 0 0 0
.86.02l2.88-1.27a3 3 0 0 1 2.43 0l2.88 1.27a1 1 0 0 0 .85-.02l3.1-1.55-.89-
1.79-1.42.71a3 3 0 0 1-2.56.06l-2.77-1.23a1 1 0 0 0-.4-.09h-.01a1 1 0 0 0-
.4.09l-2.78 1.23a3 3 0 0 1-2.56-.06l-2.3-1.15a1 1 0 0 0-.45-.11h-.01a1 1 0 0
0-.44.1l-2.21 1.11a3 3 0 0 1-2.69 0l-2.2-1.1a1 1 0 0 0-.45-.11h-.01a1 1 0 0 0-
.44.1l-2.21 1.11a3 3 0 0 1-2.69 0l-2.2-1.1a1 1 0 0 0-.45-.11h-.01zm0-2h-
4.9a21.01 21.01 0 0 1 39.61 0h-2.09l-.06-.13-.26.13h-32.31zm30.35 7.68l1.36-
.68h1.3v2h-36v-1.15l.34-.17 1.36-.68h2.59l1.36.68a3 3 0 0 0 2.69 0l1.36-
.68h2.59l1.36.68a3 3 0 0 0 2.69 0l1.36-.68h2.59l1.36.68a3 3 0 0 0
2.56.06l1.67-.74h3.23l1.67.74a3 3 0 0 0 2.56-.06zM246.18 27l16.37 4.91L278.93
27h-32.75zm-.63 2h.34l16.66 5 16.67-5h.33a3 3 0 1 1 0 6h-34a3 3 0 1 1 0-
6zm1.35 8a6 6 0 0 0 0 5.65 4h20a6 6 0 0 0 0 5.66-4H246.9z'/%3E%3Cpath d='M159.5
21.02A9 9 0 0 0 0 151 15h-42a9 9 0 0 0 0-8.5 6.02 6 6 0 0 0 .02 11.96A8.99 8.99 0
0 0 109 45h42a9 9 0 0 0 8.48-12.02 6 6 0 0 0 .02-11.96zM151 17h-42a7 7 0 0 0-
```

6.33 4h54.66a7 7 0 0 0-6.33-4zm-9.34 26a8.98 8.98 0 0 0 3.34-7h-2a7 7 0 0 1-7 7h-4.34a8.98 8.98 0 0 0 3.34-7h-2a7 7 0 0 1-7 7h-4.34a8.98 8.98 0 0 0 3.34-7h-2a7 7 0 0 1-7 7h-7a7 7 0 1 1 0-14h42a7 7 0 1 1 0 14h-9.34zM109 27a9 9 0 0 0 0-7.48 4H101a4 4 0 0 1 1 0-8h58a4 4 0 0 1 0 8h-.52a9 9 0 0 0 0-7.48-4h-42z'/%3E%3Cpath d='M39 115a8 8 0 1 0 0-16 8 8 0 0 0 0 0 16zm6-8a6 6 0 1 1-12 0 6 6 0 0 1 12 0zm-3-29v-2h8v-6H40a4 4 0 0 0 0-4 4v10H22l-1.33 4-.67 2h2.19L26 130h26l3.81-40H58l-.67-2L56 84H42v-6zm-4-4v10h2V74h8v-2h-8a2 2 0 0 0 0-2 2zm2 12h14.56l.67 2H22.77l.67-2H40zm13.8 4H24.2l3.62 38h22.36l3.62-38z'/%3E%3Cpath d='M129 92h-6v4h-6v4h-6v14h-3l.24 2 3.76 32h36l3.76-32 .24-2h-3v-14h-6v-4h-6v-4h-8zm18 22v-12h-4v4h3v8h1zm-3 0v-6h-4v6h4zm-6 6v-16h-4v19.17c1.6-.7 2.97-1.84-3.17zm-6 3.8V100h-4v23.8a10.04 10.04 0 0 0 4 0zm-6-.63V104h-4v16a10.04 10.04 0 0 0 4 3.17zm-6-9.17v-6h-4v6h4zm-6 0v-8h3v-4h-4v12h1zm27-12v-4h-4v4h3v4h1v-4zm-6 0v-8h-4v4h3v4h1zm-6-4v-4h-4v8h1v-4h3zm-6 4v-4h-4v8h1v-4h3zm7 24a12 12 0 0 0 11.83-10h7.92l-3.53 30h-32.44l-3.53-30h7.92A12 12 0 0 0 130 126z'/%3E%3Cpath d='M212 86v2h-4v-2h4zm4 0h-2v2h2v-2zm-20 0v.1a5 5 0 0 0-.56 9.65l.06.25 1.12 4.48a2 2 0 0 0 1.94 1.52h.01l7.02 24.55a2 2 0 0 0 1.92 1.45h4.98a2 2 0 0 0 1.92-1.45l7.02-24.55a2 2 0 0 0 1.95-1.52L224.5 96l.06-.25a5 5 0 0 0-.56-9.65V86a14 14 0 0 0-28 0zm4 0h6v2h-9a3 3 0 1 0 0 6H223a3 3 0 1 0 0-6H220v-2h2a12 12 0 1 0-24 0h2zm-1.44 14l-1-4h24.88l-1 4h-22.88zm8.95 26l-6.86-24h18.7l-6.86 24h-4.98zM150 242a22 22 0 1 0 0-44 22 22 0 0 0 0 44zm24-22a24 24 0 1 1-48 0 24 24 0 0 1 48 0zm-28.38 17.73l2.04-.87a6 6 0 0 1 4.68 0l2.04.87a2 2 0 0 0 2.5-.82l1.14-1.9a6 6 0 0 1 3.79-2.75l2.15-.5a2 2 0 0 0 1.54-2.12l-.19-2.2a6 6 0 0 1 1.45-4.46l1.45-1.67a2 2 0 0 0 0-2.62l-1.45-1.67a6 6 0 0 1-1.45-4.46l.2-2.2a2 2 0 0 0-1.55-2.13l-2.15-.5a6 6 0 0 1-3.8-2.75l-1.13-1.9a2 2 0 0 0-2.5-.8l-2.04.86a6 6 0 0 1-4.68 0l-2.04-.87a2 2 0 0 0-2.5.82l-1.14 1.9a6 6 0 0 1-3.79 2.75l-2.15.5a2 2 0 0 0-1.54 2.12l.19 2.2a6 6 0 0 1-1.45 4.46l-1.45 1.67a2 2 0 0 0 0 2.62l1.45 1.67a6 6 0 0 1 1.45 4.46l-.2 2.2a2 2 0 0 0 1.55 2.13l2.15.5a6 6 0 0 1 3.8 2.75l1.13 1.9a2 2 0 0 0 2.5.8zm2.82.97a4 4 0 0 1 3.12 0l2.04.87a4 4 0 0 0 4.99-1.62l1.14-1.9a4 4 0 0 1 2.53-1.84l2.15-.5a4 4 0 0 0 3.09-4.24l-.2-2.2a4 4 0 0 1 .97-2.98l1.45-1.67a4 4 0 0 0 0-5.24l-1.45-1.67a4 4 0 0 1-.97-2.97l.2-2.2a4 4 0 0 0-3.09-4.25l-2.15-.5a4 4 0 0 1-2.53-1.84l-1.14-1.9a4 4 0 0 0-5-1.62l-2.03.87a4 4 0 0 1-3.12 0l-2.04-.87a4 4 0 0 0-4.99 1.62l-1.14 1.9a4 4 0 0 1-2.53 1.84l-2.15.5a4 4 0 0 0-3.09 4.24l.2 2.2a4 4 0 0 1-.97 2.98l-1.45 1.67a4 4 0 0 0 0 5.24l1.45 1.67a4 4 0 0 1 .97 2.97l-.2 2.2a4 4 0 0 0 3.09 4.25l2.15.5a4 4 0 0 1 2.53 1.84l1.14 1.9a4 4 0 0 0 5 1.62l2.03-.87zM152 207a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm6 2a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-11 1a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-6 0a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm3-5a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-8 8a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm3 6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm0 6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm4 7a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm5-2a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm5 4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm4-6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm6-4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-4-3a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm4-3a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-5-4a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm-24 6a1 1 0 1 1 2 0 1 1 0 0 1-2 0zm16 5a5 5 0 1 0 0-10 5 5 0 0 0 0 10zm7-5a7 7 0 1 1-14 0 7 7 0 0 1 14 0zm86-29a1 1 0 0 0 0 2h2a1 1 0 0 0-2h-2zm19 9a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-14 5a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm-25 1a1 1 0 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm5 4a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm9 0a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm15 1a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm12-2a1 1 0 0 0 0 2h2a1 1 0 0

```
0 0-2h-2zm-11-14a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-19 0a1 1 0 0
0 0 2h2a1 1 0 0 0 0-2h-2zm6 5a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-
1zm-25 15c0-.47.01-.94.03-1.4a5 5 0 0 1-1.7-8 3.99 3.99 0 0 1 1.88-5.18 5 5 0
0 1 3.4-6.22 3 3 0 0 1 1.46-1.05 5 5 0 0 1 7.76-3.27A30.86 30.86 0 0 1 246
184c6.79 0 13.06 2.18 18.17 5.88a5 5 0 0 1 7.76 3.27 3 3 0 0 1 1.47 1.05 5 5 0
0 1 3.4 6.22 4 4 0 0 1 1.87 5.18 4.98 4.98 0 0 1-1.7 8c.02.46.03.93.03 1.4v1h-
62v-1zm.83-7.17a30.9 30.9 0 0 0-.62 3.57 3 3 0 0 1-.61-4.2c.37.28.78.49
1.23.63zm1.49-4.61c-.36.87-.68 1.76-.96 2.68a2 2 0 0 1-.21-3.71c.33.4.73.75
1.17 1.03zm2.32-4.54c-.54.86-1.03 1.76-1.49 2.68a3 3 0 0 1-.07-4.67 3 3 0 0 0
1.56 1.99zm1.14-1.7c.35-.5.72-.98 1.1-1.46a1 1 0 1 0-1.1 1.45zm5.34-5.77c-
1.03.86-2 1.79-2.9 2.77a3 3 0 0 0-1.11-.77 3 3 0 0 1 4-2zm42.66 2.77c-.9-.98-
1.87-1.9-2.9-2.77a3 3 0 0 1 4.01 2 3 3 0 0 0-1.1.77zm1.34 1.54c.38.48.75.96
1.1 1.45a1 1 0 1 0-1.1-1.45zm3.73 5.84c-.46-.92-.95-1.82-1.5-2.68a3 3 0 0 0
1.57-1.99 3 3 0 0 1-.07 4.67zm1.8 4.53c-.29-.9-.6-1.8-.97-2.67.44-.28.84-.63
1.17-1.03a2 2 0 0 1-.2 3.7zm1.14 5.51c-.14-1.21-.35-2.4-.62-3.57.45-.14.86-.35
1.23-.63a2.99 2.99 0 0 1-.6 4.2zM275 214a29 29 0 0 0-57.97 0h57.96zM72.33
198.12c-.21-.32-.34-.7-.34-1.12v-12h-2v12a4.01 4.01 0 0 0 7.09 2.54c.57-
.69.91-1.57.91-2.54v-12h-2v12a1.99 1.99 0 0 1-2 2 2 2 0 0 1-1.66-.88zM75
176c.38 0 .74-.04 1.1-.12a4 4 0 0 0 6.19 2.4A13.94 13.94 0 0 1 84 185v24a6 6 0
0 1-6 6h-3v9a5 5 0 1 1-10 0v-9h-3a6 6 0 0 1-6-6v-24a14 14 0 0 1 14-14 5 5 0 0
0 5 5zm-17 15v12a1.99 1.99 0 0 0 1.22 1.84 2 2 0 0 0 2.44-.72c.21-.32.34-
.7.34-1.12v-12h2v12a3.98 3.98 0 0 1-5.35 3.77 3.98 3.98 0 0 1-.65-.3V209a4 4 0
0 0 4 4h16a4 4 0 0 0 4-4v-24c.01-1.53-.23-2.88-.72-4.17-.43.1-.87.16-1.28.17a6
6 0 0 1-5.2-3 7 7 0 0 1-6.47-4.88A12 12 0 0 0 58 185v6zm9 24v9a3 3 0 1 0 6 0v-
9h-6z'/%3E%3Cpath d='M-17 191a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm19 9a1 1 0 0 1
1-1h2a1 1 0 0 1 0 2H3a1 1 0 0 1-1-1zm-14 5a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm-
25 1a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm5 4a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-2zm9
0a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm15 1a1 1 0 0 1 1-1h2a1 1 0 0
1 0 2h-2a1 1 0 0 1-1-1zm12-2a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2H4zm-11-14a1 1 0 0 1
1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-19 0a1 1 0 0 0 0 2h2a1 1 0 0 0 0-2h-
2zm6 5a1 1 0 0 1 1-1h2a1 1 0 0 1 0 2h-2a1 1 0 0 1-1-1zm-25 15c0-.47.01-.94.03-
1.4a5 5 0 0 1-1.7-8 3.99 3.99 0 0 1 1.88-5.18 5 5 0 0 1 3.4-6.22 3 3 0 0 1
1.46-1.05 5 5 0 0 1 7.76-3.27A30.86 30.86 0 0 1-14 184c6.79 0 13.06 2.18 18.17
5.88a5 5 0 0 1 7.76 3.27 3 3 0 0 1 1.47 1.05 5 5 0 0 1 3.4 6.22 4 4 0 0 1 1.87
5.18 4.98 4.98 0 0 1-1.7 8c.02.46.03.93.03 1.4v1h-62v-1zm.83-7.17a30.9 30.9 0
0 0-.62 3.57 3 3 0 0 1-.61-4.2c.37.28.78.49 1.23.63zm1.49-4.61c-.36.87-.68
1.76-.96 2.68a2 2 0 0 1-.21-3.71c.33.4.73.75 1.17 1.03zm2.32-4.54c-.54.86-1.03
1.76-1.49 2.68a3 3 0 0 1-.07-4.67 3 3 0 0 0 1.56 1.99zm1.14-1.7c.35-.5.72-.98
1.1-1.46a1 1 0 1 0-1.1 1.45zm5.34-5.77c-1.03.86-2 1.79-2.9 2.77a3 3 0 0 0-
1.11-.77 3 3 0 0 1 4-2zm42.66 2.77c-.9-.98-1.87-1.9-2.9-2.77a3 3 0 0 1 4.01 2
3 3 0 0 0-1.1.77zm1.34 1.54c.38.48.75.96 1.1 1.45a1 1 0 1 0-1.1-1.45zm3.73
5.84c-.46-.92-.95-1.82-1.5-2.68a3 3 0 0 0 1.57-1.99 3 3 0 0 1-.07 4.67zm1.8
4.53c-.29-.9-.6-1.8-.97-2.67.44-.28.84-.63 1.17-1.03a2 2 0 0 1-.2 3.7zm1.14
5.51c-.14-1.21-.35-2.4-.62-3.57.45-.14.86-.35 1.23-.63a2.99 2.99 0 0 1-.6
4.2zM15 214a29 29 0 0 0-57.97 0h57.96z'/%3E%3C/g%3E%3C/g%3E%3C/svg%3E");
  }
```

2.Creating index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Chatbot</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="{{ url_for('static',
filename='styles/style.css') }}">
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></scrip
t>
</head>

<body>
  <!-- partial:index.partial.html -->
  <section class="msger">
    <header class="msger-header">
      <div class="msger-header-title">
        <i class="fas fa-bug"></i> Chatbot <i class="fas fa-bug"></i>
      </div>
    </header>

    <main class="msger-chat">
      <div class="msg left-msg">
        <div class="msg-img" style="background-image:
url(https://image.flaticon.com/icons/svg/327/327779.svg)"></div>

        <div class="msg-bubble">
          <div class="msg-info">
            <div class="msg-info-name">Chatbot</div>
            <div class="msg-info-time">12:45</div>
          </div>

          <div class="msg-text">
            Hi, welcome to ChatBot! Go ahead and send me a message. 😄
          </div>
        </div>
      </div>

    </main>

    <form class="msger-inputarea">
```

```
      <input type="text" class="msger-input" id="textInput" placeholder="Enter
your message...">
      <button type="submit" class="msger-send-btn">Send</button>
    </form>
  </section>
  <!-- partial -->
  <script
src='https://use.fontawesome.com/releases/v5.0.13/js/all.js'></script>
  <script>

    const msgerForm = get(".msger-inputarea");
    const msgerInput = get(".msger-input");
    const msgerChat = get(".msger-chat");


    // Icons made by Freepik from www.flaticon.com
    const BOT_IMG = "https://image.flaticon.com/icons/svg/327/327779.svg";
    const PERSON_IMG = "https://image.flaticon.com/icons/svg/145/145867.svg";
    const BOT_NAME = "    ChatBot";
    const PERSON_NAME = "You";

    msgerForm.addEventListener("submit", event => {
      event.preventDefault();

      const msgText = msgerInput.value;
      if (!msgText) return;

      appendMessage(PERSON_NAME, PERSON_IMG, "right", msgText);
      msgerInput.value = "";
      botResponse(msgText);
    });

    function appendMessage(name, img, side, text) {
      //   Simple solution for small apps
      const msgHTML = `
<div class="msg ${side}-msg">
  <div class="msg-img" style="background-image: url(${img})"></div>

  <div class="msg-bubble">
    <div class="msg-info">
      <div class="msg-info-name">${name}</div>
      <div class="msg-info-time">${formatDate(new Date())}</div>
    </div>

    <div class="msg-text">${text}</div>
  </div>
</div>
`;
```

```
      msgerChat.insertAdjacentHTML("beforeend", msgHTML);
      msgerChat.scrollTop += 500;
    }

    function botResponse(rawText) {

      // Bot Response
      $.get("/get", { msg: rawText }).done(function (data) {
        console.log(rawText);
        console.log(data);
        const msgText = data;
        appendMessage(BOT_NAME, BOT_IMG, "left", msgText);

      });

    }


    // Utils
    function get(selector, root = document) {
      return root.querySelector(selector);
    }

    function formatDate(date) {
      const h = "0" + date.getHours();
      const m = "0" + date.getMinutes();

      return `${h.slice(-2)}:${m.slice(-2)}`;
    }



  </script>

</body>

</html>
```

3.Creating web using flask:

```
def chatbot_response(text):

    result, sentence = evaluate(text)
```

```
    return result

from flask import Flask, render_template, request

app = Flask(__name__)
app.static_folder = 'static'

@app.route("/")
def home():
    return render_template("index.html")

@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    return chatbot_response(userText)

if __name__ == "__main__":
    app.run()
```

Output:

```
* Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```