In [ ]:
```
import pickle
import nltk
import random
import spacy
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import os
```

In [ ]:
```
!pip install --upgrade pip

# Install spaCy
!pip install -q spacy

# Download and install the English language model
!python -m spacy download en_core_web_sm

# Install VADER
!pip3 install vaderSentiment

# Install WordNet
nltk.download('wordnet')

# Install punkt
nltk.download('punkt')

# Install stopwords
nltk.download('stopwords')
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-packages (23.1.2)
Collecting pip
  Downloading pip-24.0-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 14.1 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.1.2
    Uninstalling pip-23.1.2:
      Successfully uninstalled pip-23.1.2
Successfully installed pip-24.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the sys
tem package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Collecting en-core-web-sm==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.7.1/en_core_web_sm-3
.7.1-py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 43.6 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.8.0,>=3.7.2 in /usr/local/lib/python3.10/dist-packages (from en-core-web
-sm==3.7.1) (3.7.4)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages (from spa
cy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from spa
cy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (from spacy
<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0
,>=3.7.2->en-core-web-sm==3.7.1) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8
.0,>=3.7.2->en-core-web-sm==3.7.1) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0
,>=3.7.2->en-core-web-sm==3.7.1) (8.2.3)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (1.1.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0
,>=3.7.2->en-core-web-sm==3.7.1) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from spacy<3
.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.0.10)
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (0.3.4)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.
0,>=3.7.2->en-core-web-sm==3.7.1) (0.9.0)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (from spacy<
3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (6.4.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0
,>=3.7.2->en-core-web-sm==3.7.1) (4.66.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3
.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.31.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages
(from spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.6.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.1.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2-
>en-core-web-sm==3.7.1) (67.7.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3
```

```
.7.2->en-core-web-sm==3.7.1) (23.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3
.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.3.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7
.2->en-core-web-sm==3.7.1) (1.25.2)
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic
!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.6.0)
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/python3.10/dist-packages (from pydantic!
=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.16.3)
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydant
ic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (4.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from reques
ts<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2
.13.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0
.0,>=2.13.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0
.0,>=2.13.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2024.2.2)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.3.0,
>=8.2.2->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<
8.3.0,>=8.2.2->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.1.4)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.
0,>=0.3.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (8.1.7)
Requirement already satisfied: cloudpathlib<0.17.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from wea
sel<0.4.0,>=0.1.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3
.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.1.5)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the sys
tem package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.31.
0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from reques
ts->vaderSentiment) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSen
timent) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->va
derSentiment) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->va
derSentiment) (2024.2.2)
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 3.2 MB/s eta 0:00:00
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the sys
tem package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]:
```
True
```

In [ ]:
```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

In [ ]:
```python
with open('knowledge_base.pickle', 'rb') as handle:
  knowledge_base = pickle.load(handle)

for key in knowledge_base.keys():
  print(key, len(knowledge_base[key]))
```

```
Stardew Valley 125
ChuckleFish 11
Eric Barone 72
ConcernedApe 8
mobile 8
music 12
multiplayer 15
album 8
android 6
version 10
iOS 11
update 14
```

In [ ]:
```python
# From some text, extract possible names and randomly choose a name (if multiple names)
def get_name(name_prompt):

  nlp = spacy.load('en_core_web_sm')
```

```
  doc = nlp(name_prompt)

  # Initialize an empty list to store potential names
  names = []

  # Extract potential names
  for entity in doc.ents:
    if entity.label_ == "PERSON":
      names.append(entity.text)

  # If no "PERSON" is found then look for patterns in the sentence
  if not names:
    try:
      for token in doc:
        if token.pos_ == "PROPN" and token.dep_ != "compound":
          names.append(entity.text)
    except: # Do nothing
      pass

  if names:
    return random.choice(names)
  else:
    return ""

# Testing
# get_name("Hi there! I love stardew valley")
```

In [ ]:
```
# Return the sentiment (positive or negative) of the text

def get_sentiment(text_prompt):

  analyzer = SentimentIntensityAnalyzer()
  vs = analyzer.polarity_scores(text_prompt)
  compound_score = vs['compound']
  return compound_score

# TESTING
#text1 = "Nope"
#print(get_sentiment(text1))

#text2 = "I hate Stardew Valley."
#print(get_sentiment(text2))

#text3 = "Yes. I love stardew valley"
#print(get_sentiment(text3))

#text4 = "Not really."
#print(get_sentiment(text4))
```

In [ ]:
```
# MIGHT BE USELESS DELETE LATER

def preprocess_sentence(sentence):

  # Tokenize the raw text and lowercase
  tokenized_words = word_tokenize(sentence.lower())


  # Filter out tokens to tokens that not in the in the stopword list
  stopwords_tokens = [token for token in tokenized_words if token not in stopwords.words('english')]

  # Remove punctuation from the tokens
  punctuation_removed_tokens = [token for token in stopwords_tokens if token not in string.punctuation]

  # Lemmatize the tokens
  lemmatizer = WordNetLemmatizer()
  lemmatized_tokens = [lemmatizer.lemmatize(token) for token in punctuation_removed_tokens]

  return lemmatized_tokens

# Testing
tokens = preprocess_sentence("Is he lonely?")
#print(tokens)
#print(sentences)
```

In [ ]:
```
def calculate_similarity(sentence, word_phrase):
  # Create corpus
  corpus = [sentence, word_phrase]

  # Create vectorizer
  tfidf_vec = TfidfVectorizer()

  # Transform the corpus
  tfidf_matrix = tfidf_vec.fit_transform(corpus)

  # Calculate cosine similarity between the sentence and word vectors
  similarity = cosine_similarity(tfidf_matrix[0], tfidf_matrix[1])
```

```
  return similarity[0][0]

# Example usage
sentence1 = "I want to know more about the music of stardew valley."
word1 = "ChuckleFish"
similarity1 = calculate_similarity(sentence1, word1)
#print("Similarity:", similarity1)

sentence2 = "I want to know more about the music of stardew valley."
word2 = "Music"
similarity2 = calculate_similarity(sentence2, word2)
#print("Similarity:", similarity2)
```

In [ ]:
```
# All the keys(terms) of our knowledge base
terms_list = list(knowledge_base.keys())

# Try to determine the best term from the prompt
def get_term_from_prompt(prompt, terms_list):
  # Naive and easy way to calculate max (but this isn't an algorithms class so we good)
  similarity = 0
  similar_term = ""

  for term in terms_list:
    current_sim = calculate_similarity(prompt, term)
    if current_sim > similarity:
      similarity = current_sim
      similar_term = term

  return similar_term

# Testing
best_term = get_term_from_prompt("", terms_list)
#print(best_term)

best_term = get_term_from_prompt("Is Stardew Valley open-ended?", terms_list)
#print(best_term)
```

In [ ]:
```
# Try to determine the best sentence from a prompt
def get_sentence_from_prompt(prompt, term):
  # Naive and easy way to calculate max (but this isn't an algorithms class so we good)
  similarity = 0
  similar_sentence = ""

  try:
    sentence_list = knowledge_base[term]
    for sentence in sentence_list:
      current_sim = calculate_similarity(prompt, sentence)
      if current_sim > similarity:
        similarity = current_sim
        similar_sentence = sentence

    return similar_sentence

  except:
    return ""
# Testing
best_sentence = get_sentence_from_prompt("Is Eric Barone's alone?", "Eric Barone")
#print(best_sentence)
```

In [ ]:
```
def process_name_to_filename(name):
  filename = name.replace(" ", "_")
  #filename += ".txt"
  return filename
```

In [ ]:
```
def get_like_or_dislikes_nouns(text):
  nlp = spacy.load('en_core_web_sm')

  doc = nlp(text)

    # Initialize an empty list to store potential names
  nouns = []

  # Extract potential names
  for token in doc:
    #print(token.text, token.pos_, token.dep_)
    if token.pos_ == 'NOUN' and (token.dep_ == 'ROOT' or token.dep_ ==  'attr' or token.dep_ == 'dobj' or token
      nouns.append(token.text)
      #print(token.text, token.pos_, token.dep_)

  return nouns

# Testing

#print(get_like_or_dislikes_nouns("I like the music"))
#print(get_like_or_dislikes_nouns("I hate the music"))
#print(get_like_or_dislikes_nouns("I like the music, gameplay and the art"))
#print(get_like_or_dislikes_nouns("I like the music"))
```

In [ ]:
```
def get_like_or_dislikes_adj(text):
  nlp = spacy.load('en_core_web_sm')

  doc = nlp(text)

    # Initialize an empty list to store potential names
  jj = []

  # Extract potential names
  for token in doc:
    #print(token.text, token.pos_, token.dep_)
    if token.pos_ == 'ADJ' and (token.dep_ == 'acomp' or token.dep_ == 'conj' ):
      jj.append(token.text)
      #print(token.text, token.pos_, token.dep_)

  return jj

# Testing

#print(get_like_or_dislikes_adj("It's slow and boring."))
```

In [ ]:
```
import string

yes_synonyms = ['positive', 'yes', 'yeah', 'sure', 'certainly', 'indeed', 'absolutely', 'yup', 'ok', 'yep', 'ya
no_synonyms = ['negative', 'nay', 'nope', 'nah', 'not at all', 'never', 'no', 'not really']

def get_yes_or_no(text):
  # Lower case the text
  text = text.lower()

  # Remove punctuation
  text = ''.join(char for char in text if char not in string.punctuation)

  # Check if any entire phrase is in yes_synonyms or no_synonyms
  for phrase in yes_synonyms:
    if phrase in text:
      return 1

  for phrase in no_synonyms:
    if phrase in text:
      return -1

  # Neutral
  return 0

# TESTING
text = "not really"
response = get_yes_or_no(text)
#print(response)
```

In [ ]:
```
console_list = ['iOS', 'Android', 'Windows', 'macOS', 'Linux', 'PlayStation 4',
                'Xbox One', 'Nintendo Switch', "Playstation Vita", "PC"]

# Returns a list of consoles from a sentence
def get_consoles(sentence, console_list = ['iOS', 'android', 'Windows', 'macOS', 'Linux', 'PlayStation 4',
                'Xbox One', 'Nintendo Switch', "Playstation Vita", "PC"]):

# Check if the sentence contains any console names
  found_consoles = [console for console in console_list if console.lower() in sentence.lower()]

  return found_consoles

# Testing
#sentence = "I love playing games on my PlayStation 4 and Nintendo Switch."
#get_consoles(sentence, console_list)
```

In [ ]:
```
def get_random_fact(term):
  corpus = knowledge_base[term]
  random_sentence = random.choice(corpus)
  return random_sentence
```

In [ ]:
```
## START OF CHATBOT RULES/LOGIC

# Hard coded some greetings
greetings = [
  "Hello",
  "Greetings",
  "Pleased to meet you",
  "Hi",
  "Hey there"
]

# Hard coded some farewells
farewells = [
  "Goodbye",
  "See you later",
```

```
  "Farewells",
  "Nice talking to you"
]

#Chatbot name
chatbot_name = "Haley-Abigail"
chatbot_onscreen = chatbot_name + ": "

#Initial username (will be changed later)
user_name = "You"
user_name_onscreen = "\n" + user_name + ": "

like_set = set()
dislike_set = set()

user_model = {}

# Choose a random string from a list of strings
def choose_random(list_string):
  return random.choice(list_string)


initial_prompt = "I'm %s, a fan-bot of Stardew Valley. I love to talk about Stardew Valley! What is your name? "

user_response = input(chatbot_onscreen + choose_random(greetings) + "! " + initial_prompt)
user_name = get_name(user_response)

# Case where a name is not given/found
while not user_name or (user_name == chatbot_name):
  user_response = input(chatbot_onscreen + "Sorry about that, I couldn't get your name. Can you repeat your nam
  user_name = get_name(user_response)

# Update the username on screen
user_name_onscreen = "\n" + user_name + ": "

# Store name in the user_model
user_model["name"] = user_name

# Initialize the pickle name
user_picklename = process_name_to_filename(user_name) + ".pickle"
user_pickle_folder = 'user_models/'
user_pickle_path = user_pickle_folder + user_picklename
# Check if the folder exists
if not os.path.exists(user_pickle_folder):
  # If it doesn't exist, create it
  os.makedirs(user_pickle_folder)

# Case if the chatbot has talked this person before
if os.path.exists(user_pickle_path):
  with open(user_pickle_path, 'rb') as handle:
    stored_user_model = pickle.load(handle)
    # print(stored_user_model)
  # Get the name
  if "name" in stored_user_model:
    print(chatbot_onscreen + "Welcome back %s!" %(stored_user_model["name"]))

  # Gets if the user likes/dislikes Stardew
  if "likes_Stardew" in stored_user_model:
    # User likes stardew
    if stored_user_model["likes_Stardew"]:
      print(chatbot_onscreen + "From our last conversation, I remembered that you liked Stardew Valley.")

    # User dislikes stardew
    if not stored_user_model["likes_Stardew"]:
      print(chatbot_onscreen + "From our last conversation, I remembered that you disliked Stardew Valley.")
      response = input(chatbot_onscreen + "Do you still dislike Stardew Valley %s? %s" % (user_name, user_name_
      sentiment = get_sentiment(response)

      while not sentiment:
        # Get yes or no
        if sentiment == 0:
          sentiment = get_yes_or_no(response)

        if sentiment:
          break

        print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
        response = input(chatbot_onscreen + "Do you still dislike Stardew Valley %s? %s" % (user_name, user_nam
        sentiment = get_sentiment(response)

      # No they like Stardew now
      if sentiment < 0:
        stored_user_model["likes_Stardew"] = True

  if "likes" in stored_user_model:
    likes = stored_user_model["likes"]
    if len(likes) != 0:
      joined_likes = ' and '.join(likes)
      print(chatbot_onscreen + "When we talked last, I remembered that you liked the %s about Stardew Valley." 
```

```
    response = input(chatbot_onscreen + "Are there any additional likes about Stardew Valley since our last con
    sentiment = get_sentiment(response)

    while not sentiment:
      # Get yes or no
      if sentiment == 0:
        sentiment = get_yes_or_no(response)

      if sentiment:
        break

      print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
      response = input(chatbot_onscreen + "Are there any additional likes about Stardew Valley %s? %s" % (user_
      sentiment = get_sentiment(response)

    if sentiment > 0:
      current_likes = get_like_or_dislikes_nouns(response)

      adj_flag = False
      if not current_likes:
        current_likes = get_like_or_dislikes_adj(response)
        if current_likes:
          adj_flag = True

      # Could not process the prompt
      while not current_likes:
        response = input(chatbot_onscreen + "Sorry about that, I couldn't get your likes. Can you repeat your l
        current_likes = get_like_or_dislikes_nouns(response)
        if not current_likes:
          current_likes = get_like_or_dislikes_adj(response)
          if current_likes:
            adj_flag = True

      likes.update(current_likes)
      likes_string = " and ".join(current_likes)

      stored_user_model["likes"] = likes
      # Relay back to User
      if adj_flag:
        print(chatbot_onscreen + "You additionally like Stardew Valley because its %s." % (likes_string))
      else:
        print(chatbot_onscreen + "You additionally like %s about Stardew Valley." % (likes_string))

  if "dislikes" in stored_user_model:
    dislikes = stored_user_model["dislikes"]
    if len(dislikes) != 0:
      joined_dislikes = ' and '.join(dislikes)
      print(chatbot_onscreen + "I also remembered that you disliked the %s about Stardew Valley." %(joined_disl

    response = input(chatbot_onscreen + "Are there any additional dislikes about Stardew Valley since our last
    sentiment = get_sentiment(response)
    if sentiment == 0:
      sentiment = -1 * get_yes_or_no(response)

    while not sentiment:
      # Get yes or no
      if sentiment == 0:
        sentiment = -1 * get_yes_or_no(response)

      if sentiment:
        break

        print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
        response = input(chatbot_onscreen + "Are there any additional dislikes about Stardew Valley %s? %s" % (
        sentiment = get_sentiment(response)

    if sentiment < 0:
      current_dislikes = get_like_or_dislikes_nouns(response)

      adj_flag_dislike = False
      if not current_dislikes:
        current_dislikes = get_like_or_dislikes_adj(response)
        if current_dislikes:
          adj_flag_dislike = True

      # Could not process the prompt
      while not current_dislikes:
        response = input(chatbot_onscreen + "Sorry about that, I couldn't get your dislikes. Can you repeat you
        current_dislikes = get_like_or_dislikes_nouns(response)
        if not current_dislikes:
          current_dislikes = get_like_or_dislikes_adj(response)
          if current_dislikes:
            adj_flag_dislike = True

      dislikes.update(current_dislikes)
      dislikes_string = " and ".join(current_dislikes)

      stored_user_model["dislikes"] = dislikes
```

```
      # Relay back to User
      if adj_flag_dislike:
        print(chatbot_onscreen + "You additionally dislike Stardew Valley because its %s." % (dislikes_string))
      else:
        print(chatbot_onscreen + "You additionally dislike %s about Stardew Valley." % (dislikes_string))


  # This entire block asks if the user has played Stardew since the last time the chatbot talked
  if "played_Stardew" in stored_user_model:
    if stored_user_model["played_Stardew"]:
      try:
        platforms_played = stored_user_model["platforms_played"]
        platforms_played_string = " or ".join(platforms_played)
        question = "Have you played Stardew Valley on %s since the last time we talked %s? %s" % (platforms_pla

      except:
        question = "Have you played Stardew Valley since the last time we talked %s? %s" % (user_name, user_nam


    if not stored_user_model["played_Stardew"]:
      try:
        platforms_owned = stored_user_model["platforms_owned"]
        platforms_owned_string = " or ".join(platforms_owned)
        question = "Have you played Stardew Valley on %s since the last time we talked %s? %s" % (platforms_pla
      except:
        question = "Have you played Stardew Valley since the last time we talked %s? %s" % (user_name, user_nam

    response = input(chatbot_onscreen + question)
    sentiment = get_sentiment(response)

    while not sentiment:
      if sentiment == 0:
        sentiment = get_yes_or_no(response)

      if sentiment:
        break

      print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
      response = input(chatbot_onscreen + question)
      sentiment = get_sentiment(response)

    # Update that the user has played Stardew
    if sentiment > 0:
      stored_user_model["played_Stardew"] = True



  with open(user_pickle_path, 'wb') as handle:
    pickle.dump(stored_user_model, handle)
# ----------------------------------
# GET LIKES OR DISLIKES
# ----------------------------------
else:
  response = input(chatbot_onscreen + choose_random(greetings) + " %s. Do you like Stardew Valley? %s" % (user_
  sentiment = get_sentiment(response)

  while not sentiment:
    # Get yes or no
    if sentiment == 0:
      sentiment = get_yes_or_no(response)

    if sentiment:
      break

    print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
    response = input(chatbot_onscreen + choose_random(greetings) + " %s. Do you like Stardew Valley? %s" % (use
    sentiment = get_sentiment(response)

  # Case where the response is positive
  if sentiment > 0:
    # Store fact that the user like Stardew
    user_model["likes_Stardew"] = True

    # Ask the likes
    print(chatbot_onscreen + "I love Stardew Valley as well!")
    response = input(chatbot_onscreen + "What do you like about Stardew Valley?%s" % (user_name_onscreen))

    sentiment2 = get_sentiment(response)

    if sentiment2 > 1:
      # Get the likes of the user
      current_likes = get_like_or_dislikes_nouns(response)

      adj_flag = False
      if not current_likes:
        current_likes = get_like_or_dislikes_adj(response)
        if current_likes:
          adj_flag = True
```

```
    # Could not process the prompt
    while not current_likes:
      response = input(chatbot_onscreen + "Sorry about that, I couldn't get your likes. Can you repeat your l
      current_likes = get_like_or_dislikes_nouns(response)
      if not current_likes:
        current_likes = get_like_or_dislikes_adj(response)
        if current_likes:
          adj_flag = True

    like_set.update(current_likes)
    likes_string = " and ".join(current_likes)

    # Relay back to User
    if adj_flag:
      print(chatbot_onscreen + "You like because its %s as well? Me too! Honestly, I love everything about St
    else:
      print(chatbot_onscreen + "You like its %s as well? Me too! Honestly, I love everything about Stardew Va



# Case where the response is negative
elif sentiment < 0:

  # Store fact that the user dislikes Stardew
  user_model["likes_Stardew"] = False


  response = input(chatbot_onscreen + "Really? That's too bad. Why don't you like Stardew Valley? %s" % (user
  sentiment2 = get_sentiment(response)

  if sentiment2 < 1:
  # Get the dislikes of the user
    current_dislikes = get_like_or_dislikes_nouns(response)

    # Get adjectives if nouns not found
    adj_flag = False
    if not current_dislikes:
      current_dislikes = get_like_or_dislikes_adj(response)
      if current_dislikes:
        adj_flag = True

    # Could not process the prompt
    while not current_dislikes:
      response = input(chatbot_onscreen + "Sorry about that, I couldn't get your dislikes. Can you repeat you
      current_dislikes = get_like_or_dislikes_nouns(response)
      if not current_dislikes:
        current_dislikes = get_like_or_dislikes_adj(response)
        if current_dislikes:
          adj_flag = True


    # Update dislike set
    dislike_set.update(current_dislikes)
    dislikes_string = " and ".join(current_dislikes)

    # Relay back to User
    if adj_flag:
      print(chatbot_onscreen + "I see... You dislike because its %s." % (dislikes_string))
    else:
      print(chatbot_onscreen + "I see... You dislike its %s." % (dislikes_string))

# ------------------------------------------------------
# ASK ABOUT IF USER HAS KNOWS ConcernedApe
# ------------------------------------------------------
response = input(chatbot_onscreen + "Do you know who ConcernedApe, otherwise known as Eric Barone, is? %s" %
sentiment = get_yes_or_no(response)

while not sentiment:
  # Couldn't process prompt
  print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
  response = input(chatbot_onscreen + "Do you know who ConcernedApe, otherwise known as Eric Barone, is? %s"
  sentiment = get_yes_or_no(response)

# Yes case
if sentiment > 0:
  print(chatbot_onscreen + "Yea! He's the sole creator of Stardew Valley! Here's a random fact about him: ")
  print(chatbot_onscreen + get_random_fact("Eric Barone"))

# No case
if sentiment < 0:
  print(chatbot_onscreen + "He's the sole creator of Stardew Valley!")
  response = input(chatbot_onscreen + "What would you like to know about Eric Barone. %s" % (user_name_onscre
  fact = get_sentence_from_prompt(user_response, "Eric Barone")
  if not fact:
    fact = get_sentence_from_prompt(user_response, "ConcernedApe")
  if not fact:
    fact = get_random_fact("Eric Barone")
  print(chatbot_onscreen + fact)
```

```
# ---------------------------------
# ASK ABOUT ChuckleFish
# ---------------------------------
response = input(chatbot_onscreen + "Do you know about ChuckleFish? %s" % (user_name_onscreen))
sentiment = get_yes_or_no(response)

while not sentiment:
  # Couldn't process prompt
  print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
  response = input(chatbot_onscreen + "Do you know about ChuckleFish? %s" % (user_name_onscreen))
  sentiment = get_sentiment(response)

# Yes case
if sentiment > 0:
  print(chatbot_onscreen + "Here's a random fact about ChuckleFish: ")
  print(chatbot_onscreen + get_random_fact("ChuckleFish"))

# No case
if sentiment < 0:
  print(chatbot_onscreen + "It was the original publisher of Stardew Valley.")
  response = input(chatbot_onscreen + "What would you like to know about ChuckleFish. %s" % (user_name_onscre
  fact = get_sentence_from_prompt(response, "ChuckleFish")
  if not fact:
    fact = get_random_fact("ChuckleFish")
  print(chatbot_onscreen + fact)


# ---------------------------------------------------
# ASK ABOUT IF USER HAS EVER PLAYED STARDEW VALLEY
# ---------------------------------------------------
response = input(chatbot_onscreen + "Have you ever played Stardew Valley? %s" % (user_name_onscreen))
sentiment = get_yes_or_no(response)

while not sentiment:
  print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
  response = input(chatbot_onscreen + "Have you ever played Stardew Valley? %s" % (user_name_onscreen))
  sentiment = get_sentiment(response)

# Yes case for Played Stardew in the past
if sentiment > 0:
  # Store the fact that the user played Stardew
  user_model["played_Stardew"] = True
  response = input(chatbot_onscreen + "Which console did you play it on?%s" % (user_name_onscreen))
  # print(chatbot_onscreen + get_sentence_from_prompt(user_response, "version"))
  platforms = get_consoles(response)

  # Case when platforms is not processed
  while not platforms:
    print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
    response = input(chatbot_onscreen + "Which console did you play it on?%s" % (user_name_onscreen))
    platforms = get_consoles(response)

  # Store the platforms played
  user_model["platforms_played"] = platforms

  platforms_string = " and ".join(platforms)
  print(chatbot_onscreen + "You played on the %s platform(s)." % platforms_string)

  mobile_displayed_flag = False
  for platform in platforms:
    if platform.lower() in ("ios", "android"):
      # Makes sure that this is only processed once per for loop
      if not mobile_displayed_flag:
        print(chatbot_onscreen + "You played on Stardew Valley on mobile devices.")
        print(chatbot_onscreen + "Here's a random fact about Stardew Valley on mobile devices: ")
        print(chatbot_onscreen + get_random_fact("mobile"))
        mobile_displayed_flag = True

      print(chatbot_onscreen + "Here's fun fact about Stardew Valley on %s: " %(platform))
      print(chatbot_onscreen + get_random_fact(platform))


  # Ask if the user has played Multiplayer
  # Get yes or no
  response = input(chatbot_onscreen + "Have you ever played Stardew Valley on multiplayer? %s" % (user_name_o
  sentiment2 = get_yes_or_no(response)

  while not sentiment2:
    print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
    response2 = input(chatbot_onscreen + "Have you ever played Stardew Valley on multiplayer? %s" % (user_nam
    sentiment2 = get_sentiment(response)

  # Yes case for played multiplayer
  if sentiment2 > 0:
    user_model["played_multiplayer"] = True

  # No case for played multiplayer
  if sentiment2 < 0:
```

```
    user_model["played_multiplayer"] = False
    print(chatbot_onscreen + "It's very fun with a couple of friends!")

  print(chatbot_onscreen + "Here's a random fact about the multiplayer: ")
  print(chatbot_onscreen + get_random_fact("multiplayer"))

# No case
if sentiment < 0:
  user_model["played_Stardew"] = False
  print(chatbot_onscreen + "You should play it!")
  response = input(chatbot_onscreen + "Stardew Valley is on many platforms. Do you want to know which ones? %

  sentiment2 = get_yes_or_no(response)
  while not sentiment2:
    print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
    response = input(chatbot_onscreen + "Stardew Valley is on many platforms. Do you want to know which ones?
    sentiment2 = get_sentiment(response)

  if sentiment2 > 0:
    console_string_to_print = ', '.join(console_list[:-1]) + ', and ' + console_list[-1]
    print(chatbot_onscreen + "Stardew Valley is available on %s." %(console_string_to_print))

  response = input(chatbot_onscreen + "Do you own one of these platforms?%s" % (user_name_onscreen))
  sentiment3 = get_yes_or_no(response)

  while not sentiment3:
    print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
    response = input(chatbot_onscreen + "Do you own one of these platforms? %s" % (user_name_onscreen))
    sentiment3 = get_sentiment(response)


  if sentiment3 > 0:
    response = input(chatbot_onscreen + "Which platforms do you own?%s" % (user_name_onscreen))
    platforms = get_consoles(response)
    while not platforms:
      print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
      response = input(chatbot_onscreen + "Which platforms do you own?%s" % (user_name_onscreen))
      platforms = get_consoles(response)

    # Store the platforms played
    user_model["platforms_owned"] = platforms
    user_platforms_owned = ' or '.join(platforms)
    print(chatbot_onscreen + "You should play Stardew Valley on %s." %(user_platforms_owned))


# ------------------------------------------------------
# ASK ABOUT IF USER HAS LIKES THE MUSIC
# ------------------------------------------------------
response = input(chatbot_onscreen + "Do you like the music of Stardew Valley? %s" % (user_name_onscreen))
sentiment = get_sentiment(response)

while not sentiment:
  # Get yes or no
  if sentiment == 0:
    sentiment = get_yes_or_no(response)

  if sentiment:
    break

  print(chatbot_onscreen + "Sorry about that, I couldn't get your response.")
  response = input(chatbot_onscreen + "Do you like the music of Stardew Valley? %s" % (user_name_onscreen))
  sentiment = get_sentiment(response)

# Yes case
if sentiment > 0:
  # Store the fact the user likes the music
  like_set.add("music")


  print(chatbot_onscreen + "Here's a random fact about its music: ")
  print(chatbot_onscreen + get_random_fact("music"))

  response = input(chatbot_onscreen + "Did you know that Stardew Valley also has an album?%s" % (user_name_on

  #response doesn't matter here (its fake)
  print(chatbot_onscreen + get_random_fact("album"))



# No case
if sentiment < 0:
  # Store the fact the user dislikes the music
  dislike_set.add("music")
  print(chatbot_onscreen + "You should give it another chance!")
  print(chatbot_onscreen + "They even have a physical vinyl album!")

# Adding likes/dislikes to the user model
user_model["likes"] = like_set
user_model["dislikes"] = dislike_set
```

```
  with open(user_pickle_path, 'wb') as handle:
    pickle.dump(user_model, handle)
```

```
Haley-Abigail: Greetings! I'm Haley-Abigail, a fan-bot of Stardew Valley. I love to talk about Stardew Valley!
What is your name?
You: Hi there Haley-Abigail
Haley-Abigail: Sorry about that, I couldn't get your name. Can you repeat your name please?
You: My name is John
Haley-Abigail: Pleased to meet you John. Do you like Stardew Valley?
John: Yes
Haley-Abigail: I love Stardew Valley as well!
Haley-Abigail: What do you like about Stardew Valley?
John: I don't like the music
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-25-065713908a28> in <cell line: 64>()
    346   # ASK ABOUT IF USER HAS KNOWS ConcernedApe
    347   # -----------------------------------------------------
--> 348   response = input(chatbot_onscreen + "Do you know who ConcernedApe, otherwise known as Eric Barone, is
? %s" % (user_name_onscreen))
    349   sentiment = get_yes_or_no(response)
    350

/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in raw_input(self, prompt)
    849                 "raw_input was called, but this frontend does not support input requests."
    850             )
--> 851         return self._input_request(str(prompt),
    852             self._parent_ident,
    853             self._parent_header,

/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent,
password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from None
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

In [ ]:

```
# Ran out of conversation topics, will repeat this until the end.

print(chatbot_onscreen + "I have ran out of questions for you." )
print(chatbot_onscreen + "Now it's your turn to ask!")
print(chatbot_onscreen + "These are the following terms I know alot about: " )
print(chatbot_onscreen + "Type in 'bye' to end this conversation" )

# Printing the terms
for term in terms_list:
  print("  * " + term)

while(True):
  user_response = input(chatbot_onscreen + "What would you like to know about one of the terms?%s" % (user_name
  if user_response.lower() == ('bye' or 'goodbye'):
    break
  term_from_user = get_term_from_prompt(user_response, terms_list)
  if not term_from_user:
    term_from_user = "Stardew Valley"

  answer = get_sentence_from_prompt(user_response, term_from_user)
  if not answer:
    print(chatbot_onscreen + "Sorry. I couldn't understand your question")
  else:
    print(chatbot_onscreen + answer)

print(chatbot_onscreen + choose_random(farewells) + ".")
```