```python
import os
import re
import nltk
import pickle
import random
import knowledge_base #file containing all processed data
# from textblob import TextBlob
from autocorrect import Speller
# from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.util import ngrams
from nltk.stem import WordNetLemmatizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer

#creating an object of sentiment intensity analyzer
sia= SentimentIntensityAnalyzer()
spell = Speller()
wnl = WordNetLemmatizer()

def retrieve_data():
    """Collects the existing data from the pickle files and other files with functions to parse through
corpus text files.

    Returns:
        tuple: Returns the tuple containing movie_dictionary, tvshow_dictionary, freq_words, dictionary of
all documents
        in a string and the vocabulary.
    """
    with open("./Corpuses/movies.pickle", "rb") as movie_file:
        movies_dict = pickle.load(movie_file)

    with open("./Corpuses/tvshows.pickle", "rb") as tv_file:
        tvshows_dict = pickle.load(tv_file)
    # movies_dict, tvshows_dict = knowledge_base.filmography()

    frequent_words_list, vocab = knowledge_base.tfidf()

    list_of_docs = knowledge_base.docsentokens()

    return movies_dict, tvshows_dict, frequent_words_list, list_of_docs, vocab

# -----------------------------------------------------------------------------------------------------
----------------------------------------

# Some global variables with few hardcoded responses.

movies, tvshows, freq_words, doc_dict, vocabulary = retrieve_data()

pronunciation = '/kiˈɑːnuː/ kee-AH-noo'
stop_words = set(stopwords.words('english'))

query_words = ['who', 'what', 'when', 'how', 'which', 'why', 'where']
positive_responses = ["That sounds great!", "You are absolutely correct!", "I agree with that!", "Yes that
is true."]

negative_responses = ["Whoa let's cool down.", "I understand, but let's be positive!", "That is a bit
extreme, we can always say something good."]

# abbreviations = {'idk' : "I don't know",
#                  'btw' : "by the way",
#                  'ok' : 'okay',
#                  'asap' : 'as soon as possible',
#                  'lmk' : "let me know",
#                  'imo' : "in my opinion",
#                  'nvm' : "never mind",
#                  'aka' : 'also known as'}


response_dict = {'hello': "Hello I am KeanuBot! You can ask me any questions about Keanu Reeves!",
                 'Goodbye': 'Adios! Happy Speeding! (I meant the movie, not rash driving)',
                 'what is your name': 'My name is KeanuBot. I am a simple chatbot. You can ask me any
questions about Keanu Reeves!',
                 'how are you': 'I am fine! But this is not about me, its about Keanu!',
                 'who_is' : "Keanu Charles Reeves (/kiˈɑːnuː/ kee-AH-noo; born September 2, 1964) is a 
Canadian[c] actor. Born in Beirut and raised in Toronto, he made his acting debut in the Canadian 
television series Hangin' In (1984), before making his feature film debut in Youngblood (1986). Reeves had 
his breakthrough role in the science fiction comedy Bill & Ted's Excellent Adventure (1989), and he 
reprised his role in its sequels.",
                 'born': "Keanu's birthday is on 2nd September, 1964. He was born in Beirut and raised in 
Toronto.",
                 'mother': "Keanu is the son of Patricia (née Taylor), a costume designer and performer, 
and Samuel Nowlin Reeves Jr. His mother is English, originating from Essex.",
                 'what you do': "I am KeanuBot, I answer queries about him, and do not know anything 
else."
                }

# ----------------------------------------------------------------------------------------------------
--------------------------------------------

def preprocess_text(text):
    """Function to preprocess text string by tokenizing, removing stopwords and removing non-alphanumeric 
words.

    Args:
        text (string): The text string to be preprocessed.

    Returns:
        string: returns the preprocessed text
    """
    tokens = nltk.word_tokenize(text.lower())
    tokens = [(token) for token in tokens if token not in stop_words and token.isalnum()]
    return ' '.join(tokens), tokens

# ----------------------------------------------------------------------------------------------------
--------------------------------------------

def cosim(user_response,sentence):
    """Function to calculate similarity between two given strings.

    Args:
        user_response (string): The user query.
        sentence (string): One sentence extracted from the knowledge base.

    Returns:
        float: The cosine_similarity value with range between 0.0 and 1.0.
    """

    tokens = nltk.word_tokenize(sentence)
    tokens = [word.lower() for word in tokens if word.isalnum() and word!='keanu' and word!='reeves']
    tokens = [word for word in tokens if word not in stop_words]

    user_tokens = nltk.word_tokenize(user_response)
    user_tokens = [word.lower() for word in user_tokens if word.isalnum() and word!='keanu' and 
word!='reeves']
    user_tokens = [word for word in user_tokens if word not in stop_words]

    preproc_sent = " ".join(tokens)
    preproc_user = " ".join(user_tokens)

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([preproc_sent, preproc_user])

    # Calculate cosine similarity
    cosine_sim = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])

    return cosine_sim[0][0]

# ----------------------------------------------------------------------------------------------------
-------------------------------

def get_sentences(query):
    """Function to parse through knowledge base and return sentences based on cosine similarity between 
the
        user query and knowledge base sentences.

    Args:
        query (string): The user query string.
    Returns:
        list: List of sentences for response output.
    """
    tokens = nltk.word_tokenize(query)

    # bigrams = ngrams(tokens, 2)
    # tokens = []
    tokens = [word.lower() for word in tokens if word.isalnum() and word!='keanu' and word!='reeves' and
word!='movie']
    if len(tokens) == 0 :
    # or tokens == []:
        return None
    sentences = []
    query_tokens = nltk.word_tokenize(query.lower())
    q_tokens = [tokens for tokens in query_tokens if tokens.isalnum() and tokens not in stop_words and
tokens!='keanu' and tokens!='reeves' and tokens!='movie']
    for i in doc_dict.keys():
        if len(sentences) >= 5:
            break
        for j in doc_dict[i]:
            if (cosim(query, j) > 0.2) or any(word in j for word in q_tokens):
                if len(sentences) < 5 and len(j) > 20:
                    sentences.append(j)
                else:
                    break
    if len(sentences) == 0:
        return None
    else:
        return sentences

# -----------------------------------------------------------------------------------------------------
-------------------------------------------

def movies_print(user_year, flag):
    """Function to print movies based on the user's birth year the first time and random values next time
onwards.

    Args:
        user_year (string): The year of birth as entered by the user.
        flag (int): Flag value to specify is the function should return movies of user's birth year or
else random years.

    Returns:
        tuple: Tuple containing the year (key) and the movies list (values).
    """
    if user_year in movies.keys() and flag == 0:
        return user_year, movies[user_year]
    else:
        random_key = random.choice(list(movies.keys()))
        return random_key, movies[random_key]

# -----------------------------------------------------------------------------------------------------
-------------------------------------------

def tvshows_print(user_year, flag):
    """Function to print tvshows based on the user's birth year the first time and random values next time
onwards.

    Args:
        user_year (string): The year of birth as entered by the user.
        flag (int): Flag value to specify is the function should return tvshows of user's birth year or
else random years.

    Returns:
        tuple: Tuple containing the year (key) and the tvshows list (values).
    """
    if user_year in tvshows.keys() and flag == 0:
        return user_year, tvshows[user_year]
    else:
        random_key = random.choice(list(tvshows.keys()))
        return random_key, tvshows[random_key]

# -----------------------------------------------------------------------------------------------------
-------------------------------------------
```

```python
def response_parse(user_name, user_year):
    """Function to get query from the user and return the relevant response.

    Args:
        user_name (string): The name of the user.
        user_year (string): The year of birth as entered by the user.
    """
    cont_conv = True
    query = ''
    movie_flag, tvshow_flag = 0, 0
    while(cont_conv == True):
        query = input("\nAsk Anything: ")
        query = spell(query)
        query = re.sub('means', 'keanu', query)
        query = query.lower()
        query = re.sub('mom', 'mother', query)
        query = re.sub('dad', 'father', query)
        query = re.sub('birthday', 'born', query)
        # query = re.sub('birthday', 'born', query)
        # print(query)
        sentiment = sia.polarity_scores(query)

        if query.lower() == 'q':
            print("\nIt was great meeting you! KeanuSpeed.\n\n")
            cont_conv = False
            exit(1)

        if sentiment['pos'] > 0.4 and not any(word for word in query_words if word in query):
            random_pos = random.choice(positive_responses)
            print(random_pos)

        elif sentiment['neg'] > 0.3 and not any(word for word in query_words if word in query):
            random_neg = random.choice(negative_responses)
            print(random_neg)

        elif 'how are you' in query.lower():
            print('\n', user_name, ", ", response_dict['how are you'])

        elif 'your name' in query.lower():
            print('\n', response_dict['what is your name'])

        elif 'who' in query.lower() and 'you' in query.lower():
            print('\n', response_dict['what is your name'])

        elif 'pronunc' in query.lower():
            print('\n', 'Keanu is pronounced as' + pronunciation)

        elif 'what' in query.lower() and 'you' in query.lower() and 'do' in query.lower():
            print('\n', response_dict['what you do'])

        # elif 'mother' in query.lower() or 'mom' in query.lower():
        #     print('\n', response_dict['mother'])

        elif ('who is he' in query.lower()) or (query.lower() == 'who is keanu?') or (query.lower()== 'who
is keanu reeves?'):
            print('\n', response_dict['who_is'])

        # elif 'born' in query.lower() or 'birthday' in query.lower():
        #     print('\n', response_dict['born'])

        elif 'movies' in query.lower():
            print("\nHere are some movies Keanu has acted in:")
            yearm, movie = (movies_print(user_year, movie_flag))
            print("In the year", yearm, ", These are the tvshows Keanu starred in:", movie)
            movie_flag = 1

        elif 'tvshows' in query.lower() or 'tv shows' in query.lower():
            print('\n', "Here are some tv shows Keanu has acted in:")
            yeart, tvshow = (tvshows_print(user_year, tvshow_flag))
            print("In the year", yeart, ", These are the tvshows Keanu starred in:", tvshow)
            tvshow_flag = 1

        else:
            sent = get_sentences(query.lower())
            if sent == None:
                print('\n', "I'm not sure I understand that", user_name, ", please ask something else.")
            else:
                print('\n', " ".join(sent))

# ------------------------------------------------------------------------------------------------------
--------------------------------------------

def create_user_def():
    """Function to create a user and ask for user details, and then store it in a text file (for easy
viewability)
        and a pickle file (for easy accessability).
    """
    import pickle
    cont_convo = True
    user_dict = {}
    user_type = 'new'


print("|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||")

    print("Hello! I am KeanuBot. Here to help you with anything regarding the Hollywood Actor Keanu
Reeves.")
    print(" Please be patient as I am not the perfect bot, I am not even a little bit close to GPT model.")
    print("     Due to my limited abilities, I would present irrelevant answers, please ignore them.")
    print("            You can ask about Keanu's movies, personal life, facts or such.")
    print("\n             If you want to quit the conversation, press 'q' and enter")

print("|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||")


    while(cont_convo == True):

        user_input = input("\n\nMay I know your name, please: ")
        user_input = user_input.title()
        user_input = re.sub(' ', '', user_input)
        if user_input == 'q':
            print("It was great meeting you! KeanuSpeed.")
            cont_convo = False
            exit(1)
        elif os.path.exists('./usermodels/' + user_input + '.txt'):
            user_type = 'returning'
            with open('./usermodels/' + user_input + '.pickle', 'rb') as userpickle:
                user_dict = pickle.load(userpickle)
            print("Welcome back " + user_input + "! Its good to see you again!")
        else:
            print("Hello! " + user_input + ". Its great to meet you.")
            user_dict['name'] = user_input
            choice = 'y'
            while choice.lower() == "y":
                user_input = input("\nWhat is your favorite colour?:")
                user_input = spell(user_input)
                if user_input.lower() == 'q':
                    print("\nIt was great meeting you! KeanuSpeed.")
                    cont_convo = False
                    exit(1)
                if user_input.lower() == 'green':
                    print("Keanu's favourite color is green too!")
                    choice = 'n'
                else:
                    print("Ooff that's close, would've been great if it were Green because that's Keanu's
favourite color.")
                    print("Do you want to change your answer?")
                    choice = input("Enter y/n ")

            user_dict['colour'] = user_input

            flag = True
            while(flag == True):
                user_input = input("\nWhich year were you born in? ")
                if user_input == 'q':
                    print("It was great meeting you! KeanuSpeed.")
                    cont_convo = False
                    exit(1)
                if not user_input.isnumeric() or not len(user_input)==4:
                    print('Enter proper year')
```

```python
                else:
                    flag = False
            user_dict['year'] = user_input

        if user_type == 'new':
            pickle_file_name = './usermodels/'+ user_dict['name'] + '.pickle'
            file_name = './usermodels/'+ user_dict['name'] + '.txt'
            with open(file_name, 'w') as file:
                for k,v in user_dict.items():
                    file.write(str(k) + " : " + str(v) + "\n")

            with open(pickle_file_name, 'wb') as handle:
                pickle.dump(user_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)

        cont_convo = False

    print("You can ask any question about Keanu now, his birthday, parents, movies etc...")

    response_parse(user_dict['name'], user_dict['year'])

# ----------------------------------------------------------------------------------------------------
--------------------------------------------

create_user_def()
```