Gradio UI - 21 May 2023 – My Experimentation and learning

Gradio is an open-source library that was developed by Abubakar Abid, the CEO and Co-founder of Gradio.ai. It was created to simplify the process of creating user interfaces for machine learning models.
While Gradio can be used for a wide range of machine learning applications, including chatbots, it is not specifically designed exclusively for chatbots. Gradio provides a simple and intuitive way to create interactive interfaces for various types of models, including text, image, audio, and video models. It allows users to easily input data and visualize the model's outputs.
Gradio's flexibility makes it suitable for a variety of use cases, including chatbots, sentiment analysis, image classification, style transfer, question answering systems, and more. Its goal is to provide a user-friendly and accessible way for developers and researchers to share and demonstrate their machine learning models.
Whether you are building a chatbot or any other type of interactive machine learning application, Gradio can be a valuable tool to quickly prototype, test, and showcase your models with a user-friendly interface.

Topic	Description
What is Gradio?	Gradio is an open-source Python library that allows you to create user interfaces for machine learning models in a web-based environment.
Language	Gradio is primarily written in Python.
Usage	Gradio can be used for rapid prototyping, model testing, debugging, demonstrating models, collecting feedback, and integration into Python notebooks or webpages.
Interface	Gradio interfaces can be created for text, image, audio, and video models across different platforms such as TensorFlow, PyTorch, etc.
Execution Environment	Gradio applications can run both locally and on the web, but in both cases, the interface is presented in a web browser.
Local/Desktop Execution	Gradio code can be run on your local machine. By default, launching a Gradio interface starts a local server that you can access from your web browser.
Online/Web Execution	The same Gradio code can also be run online. If you want your Gradio interface to be accessible over the internet, use the `share=True` parameter when you launch the interface.
Web Framework	Gradio uses Flask, a micro web framework written in Python, to create a local web server for the interface.
Sharing	Gradio interfaces can be shared with others over the internet using the `share=True` option.
Security	When exposing a local server to the internet, ensure you are following good security practices.
Alternatives to Gradio	Streamlit, Dash by Plotly, Jupyter Widgets (ipywidgets), Voila, Panel by HoloViz, Flask, Django, PyQt, Tkinter.
Easiest to Start With	Streamlit and Gradio are considered the most beginner-friendly and quickest for prototyping.
Gradio vs PyQt	Gradio simplifies creating interfaces for machine learning models by providing high-level abstractions, seamless integration with ML libraries, built-in model sharing, and a user-friendly, web-based approach. In comparison, PyQt, a

	traditional GUI tool, requires more manual setup and programming for similar use-cases.
Gradio Execution	Gradio does not use desktop windowing systems like PyQt or Tkinter, it runs a local server (using Flask) which is accessed via a web browser. This applies whether running locally or over the internet.
Gradio and Notebooks	Gradio can be easily integrated into Jupyter notebooks, allowing you to create and view interactive interfaces within the notebook environment. This can be useful for exploring and demonstrating models during development.
Gradio and Hugging Face Spaces	Hugging Face Spaces is a platform for hosting and sharing machine learning models. It allows you to create web applications using frameworks like Gradio and Streamlit. With Gradio, you can create user-friendly interfaces for your Hugging Face models and host them in Spaces.

Converting the provided PyQt code to Gradio involves shifting the focus from event-driven GUI programming (as PyQt does) to functional programming which Gradio is based on. Here's a Gradio implementation of your code. It accepts an uploaded CSV file, performs the same analyses, and returns the modified CSV file as a downloadable output. Note that for the purpose of simplicity and readability, this script assumes the presence of a column named 'Content' in the input CSV.

```
import pandas as pd
import spacy
import gradio as gr
import csv
from nrclex import NRCLex
from transformers import pipeline
from rake_nltk import Rake

# Initialize objects
emotion_pipeline = pipeline('sentiment-analysis', model='nlptown/bert-base-multilingual-uncased-
sentiment')
nlp = spacy.load('en_core_web_sm')
rake = Rake()

def process_csv(file):
    reader = csv.DictReader(file)
    emotions = []
    sentiments = []
    entities = []
    keywords = []
    for row in reader:
        text = row['Content']  # Replace 'Content' with the correct column name
        nrc_obj = NRCLex(text)
        emotion_scores = nrc_obj.affect_frequencies
        emotions.append(emotion_scores)
        sentiment = analyze_emotion(text)
        sentiments.append(sentiment)
        entities.append(analyze_entities(text))
        keywords.append(extract_keywords(text))  # Extract keywords for each text

    fieldnames = reader.fieldnames + list(emotions[0].keys()) + ['sentiment', 'entities', 'keywords']
    output = []
    for row, emotion_scores, sentiment, entity, keyword in zip(reader, emotions, sentiments, entities,
keywords):
        row.update(emotion_scores)  # Update the row dictionary with emotion scores
```

```
        row.update({'sentiment': sentiment, 'entities': entity, 'keywords': keyword})  # Update the
row dictionary with sentiment, entities and keywords
        output.append({field: row.get(field, '') for field in fieldnames})  # Write row with matching
fields or empty values
    return pd.DataFrame(output).to_csv(index=False)

def analyze_emotion(text):
    result = emotion_pipeline(text)[0]
    sentiment = result['label']
    return sentiment

def analyze_entities(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities

def extract_keywords(text):
    rake.extract_keywords_from_text(text)
    return rake.get_ranked_phrases()  # Extract keywords from text

iface = gr.Interface(fn=process_csv, inputs=gr.inputs.File(type='csv'), outputs=gr.outputs.File())
iface.launch()
```

Enhanced:

```
import pandas as pd
import spacy
import gradio as gr
import csv
from nrclex import NRCLex
from transformers import pipeline
from rake_nltk import Rake

# Initialize objects
emotion_pipeline = pipeline('sentiment-analysis', model='nlptown/bert-base-multilingual-uncased-
sentiment')
nlp = spacy.load('en_core_web_sm')
rake = Rake()

def process_csv(file):
    reader = csv.DictReader(file)
    emotions = []
    sentiments = []
    entities = []
    keywords = []
    for row in reader:
        text = row['Content']
        nrc_obj = NRCLex(text)
        emotion_scores = nrc_obj.affect_frequencies
        emotions.append(emotion_scores)
        sentiment = analyze_emotion(text)
        sentiments.append(sentiment)
        entities.append(analyze_entities(text))
        keywords.append(extract_keywords(text))

    fieldnames = reader.fieldnames + list(emotions[0].keys()) + ['sentiment', 'entities', 'keywords']
    output = []
    for row, emotion_scores, sentiment, entity, keyword in zip(reader, emotions, sentiments, entities,
keywords):
        row.update(emotion_scores)
        row.update({'sentiment': sentiment, 'entities': entity, 'keywords': keyword})
        output.append({field: row.get(field, '') for field in fieldnames})
    df = pd.DataFrame(output)
    return df, df.to_csv(index=False)

def analyze_emotion(text):
```

```
    result = emotion_pipeline(text)[0]
    sentiment = result['label']
    return sentiment

def analyze_entities(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities

def extract_keywords(text):
    rake.extract_keywords_from_text(text)
    return rake.get_ranked_phrases()

def instructions():
    return "Please upload a CSV file. The file should contain a column named 'Content'. After
processing, a preview of the results will be shown and a new CSV file with additional columns for
sentiment, entities, and keywords can be downloaded."

iface = gr.Interface(fn=process_csv,
                     inputs=["csv", instructions],
                     outputs=["dataframe", gr.outputs.File(extension=".csv")],
                     title="Emotion and Keyword Analyzer",
                     description="Upload a CSV file that contains a 'Content' column. The content will
be analyzed for sentiment, entities, and keywords.")
iface.launch()
```