



## UNIVERSITÀ DEGLI STUDI DI CAGLIARI

*Corso di laurea in  
Data Science Business Analytics e Innovazione*

**Sentiment Analysis, Emotion Analysis e Topic Modeling del podcast One More Time di Luca Casadei: dall'audio testo dei video ai commenti degli utenti su YouTube**

**Report tecnico del progetto per l'esame del corso di  
Web Analytics e Analisi Testuale**

Anno Accademico 2023/2024

a cura di

***Mariella Ruiu, Matricola N. 11/82/00362***

# INDICE

INTRODUZIONE PROGETTO	3
ESECUZIONE PROGETTO	4
PRIMA PARTE DEL PROGETTO: ANALISI DELL' AUDIO TESTO DI VIDEO ESTRATTI DA YOUTUBE	5
1) <i>Selezione di due video tra i più popolari del canale One More Time su YouTube</i>	5
2) <i>Creazione di uno script per l'estrazione dell'audio da un video YouTube</i>	5
3) <i>Conversione da file audio (formato .mp3) a testo (formato .txt)</i>	7
4) <i>Analisi del sentiment e delle emozioni sugli audio testo delle interviste</i>	8
5) <i>Visualizzazioni grafiche: analisi di sentiment ed emozioni sull' audio testo delle interviste</i>	11
6) <i>Analisi dei topic rilevanti sulle frasi dei testi delle interviste</i>	16
7) <i>Analisi comparativa sul sentiment, sulle emozioni e sulla coerenza dei topic dei testi delle interviste</i>	24
8) <i>Conclusioni sulla prima parte del progetto</i>	26
SECONDA PARTE DEL PROGETTO: ANALISI DEI COMMENTI ESTRATTI DA YOUTUBE	27
1) <i>Selezione di dieci video tra i più popolari del canale One More Time su YouTube</i>	27
2) <i>Creazione di uno script per l'estrazione dei commenti di video di YouTube</i>	27
3) <i>Analisi del sentiment e delle emozioni sui commenti delle interviste</i>	30
4) <i>Creazione grafici sul sentiment e sulle emozioni, e analisi comparativa dei commenti delle interviste</i>	32
5) <i>Analisi dei topic sui commenti delle interviste di One More Time su YouTube</i>	38
6) <i>Conclusioni sulla seconda parte del progetto</i>	49

## **Sentiment Analysis, Emotion Analysis e Topic Modeling del podcast One More Time di Luca Casadei: dall'audio testo dei video ai commenti degli utenti su YouTube**

### **Introduzione progetto**

One More Time<sup>1</sup> è uno dei podcast più seguiti in Italia, ideato e condotto da Luca Casadei. Si tratta di un format basato su delle interviste in cui gli intervistati sono personaggi conosciuti, provenienti da varie realtà, che raccontano il loro percorso di vita, focalizzandosi su delle esperienze significative. YouTube è una delle principali piattaforme in cui viene mandato in onda il podcast, attualmente conta più di trecentomila iscritti e milioni di visualizzazioni alle spalle.

Il progetto sarà diviso in due parti differenti e tratterà come argomento il canale YouTube appena descritto.

#### **1. Nella prima parte del progetto verrà effettuata un'analisi dell'audio testo di video estratti da YouTube:**

- verranno selezionati due video tra i più popolari del canale;
- sarà estratto il contenuto audio dai video selezionati utilizzando tecniche di web scraping;
- l'audio dei video verrà trascritto in formato testuale attraverso strumenti di speech-to-text;
- verrà effettuata la sentiment analysis (positivo, negativo) non sul testo complessivo, ma su ogni singola frase;
- verrà effettuata l'emotion analysis (es. gioia, paura, rabbia) non sul testo complessivo, ma su ogni singola frase;
- ogni video segue una sequenza temporale, quindi per ogni frase dell' audio testo del video verrà effettuata un'analisi di come cambia sia il sentiment che l'emotion;
- per ogni video, verranno applicate tecniche di topic modeling sul testo dell'intervista, con l'estrazione massima di 5 topic in modo tale che vengano individuati gli argomenti chiave;
- infine, verrà effettuata un'analisi comparativa tra i video scelti, confrontando sentiment, emozioni e topic emersi.

#### **2. Nella seconda parte del progetto verrà effettuata un'analisi dei commenti estratti da video su YouTube:**

- verranno selezionati dieci tra i video i più popolari del canale;
- saranno estratti i commenti sotto ogni video selezionato utilizzando tecniche di web scraping;
- verrà effettuata la sentiment analysis (positivo, negativo) sui commenti estratti;
- verrà effettuata l'emotion analysis (es. gioia, paura, rabbia) su ogni singolo commento;
- per ogni video, verranno applicate tecniche di topic modeling sui commenti, con l'estrazione massima di 5 topic in modo tale che vengano individuati gli argomenti chiave;
- infine, verrà effettuata un'analisi comparativa tra i commenti dei video selezionati, confrontando sentiment, emotion e topic emersi.

---

<sup>1</sup> Link canale One More Time di Luca Casadei <https://www.youtube.com/@OneMoreTimePodcast>

## Esecuzione progetto

Per eseguire il procedimento del progetto verrà utilizzato il linguaggio di programmazione Python<sup>2</sup> e un ambiente di sviluppo integrato (IDE) sviluppato da JetBrains chiamato PyCharm<sup>3</sup>, utile per scrivere codice Python in modo efficiente.

Per eseguire adeguatamente entrambe le parti del progetto, verranno creati due branch distinti, ognuno dei quali rappresenta una parte del progetto. Per lo sviluppo di esso sarà creato un ambiente virtuale isolato che consentirà di gestire in modo appropriato le dipendenze delle librerie, in modo tale che non si presentino conflitti e confusione con il materiale esterno. Dopo aver creato l'ambiente virtuale, si creerà un file standard "requirements.txt", il quale permetterà di elencare le dipendenze Python necessarie.

Il file requirements.txt verrà copiato nella directory principale della parte di progetto presa in considerazione in modo tale che il file con le dipendenze Python sia facilmente accessibile e visibile.

Nel report del progetto saranno presentati blocchi di codice dei vari script per facilitare la comprensione dello sviluppo del progetto. Ogni parte del progetto seguirà un ordine coerente con la descrizione fornita. Le librerie utilizzate per ogni script saranno indicate al suo interno. Oltre ai blocchi di codice e ai messaggi visualizzati a schermo, nel report saranno presenti i grafici delle analisi del progetto, che verranno numerati.

Nel branch **main** creato direttamente sulla piattaforma Bitbucket, verranno inseriti, oltre al testo introduttivo del progetto README, anche il report tecnico e la presentazione PowerPoint.

---

<sup>2</sup> Link per documentazione linguaggio programmazione Python [Our Documentation | Python.org](#)

<sup>3</sup> Link sito di PyCharm [PyCharm: l'IDE Python per la scienza dei dati e lo sviluppo web \(jetbrains.com\)](#)

## **Prima parte del progetto: analisi dell' audio testo di video estratti da YouTube**

La prima parte del progetto si occupa di effettuare un'analisi sull'audio testo di video su YouTube, e nel branch chiamato "**analisi\_AudioTesto\_videoYouTube**" vengono raccolti dati e script necessari per effettuare: sentiment analysis, emotion analysis, topic modeling e analisi comparativa tra i video delle interviste di "One More Time" selezionate. Verranno spiegati i passaggi affrontati per completare la prima parte del progetto.

### **1) Selezione di due video tra i più popolari del canale One More Time su YouTube**

Per la prima parte del progetto, ho selezionato due tra i video più popolari del canale di *One More Time* di Luca Casadei. Ho scelto di concentrarmi sui video di Steven Basalari e Bianca Balti come ospiti, in quanto entrambi hanno avuto un discreto successo e rispettivamente vanno oltre le 973.000 e 868.000 visualizzazioni.

### **2) Creazione di uno script per l'estrazione dell'audio da un video YouTube**

Nel file **YouTubeAudioScraper.py** è stato scritto un codice che permette l'estrazione dell'audio in formato MP3 da video su YouTube. Il codice presente nello script segue le regole dell'articolo "How to Download a YouTube Video in MP3 Format with Python" di Salim Oyinlola<sup>4</sup>. Il codice presente nell'articolo è stato modificato per poter usare la programmazione ad oggetti, per evitare di ripetere lo stesso codice per ogni intervista.

Per effettuare lo scaricamento degli audio da YouTube mi sono servita principalmente della libreria **pytube** che consente il processo di download di video da YouTube.

```
class DownloadAudioYouTube:  
    """Creazione classe DownloadAudioYouTube usata per effettuare  
    i download di audio da YouTube."""  
  
    def __init__(self, URL):  
        """Metodo costruttore richiamato ogni volta che viene creato un nuovo oggetto della classe,  
        inizializza l'attributo URL con il link del video di YouTube da cui verrà scaricato l'audio"""  
  
        self.URL = URL
```

Ho creato la classe **DownloadAudioYouTube** per effettuare il download diretto dell'audio di un video su YouTube in formato mp3 per ciascuna istanza creata che verrà definita in seguito.

Nel **costruttore** ho inserito l'attributo URL che si riferisce al link del video su YouTube. Questo verrà richiamato in modo diverso per ogni intervista quando verrà creata l'istanza tramite la creazione di un oggetto.

---

<sup>4</sup> Link articolo per scaricare audio mp3 da YouTube: <https://code.pieces.app/blog/how-to-download-a-youtube-video-in-mp3-format-with-python>

```

def download(self):
    """Metodo che effettua il download dal video su YouTube tramite l'URL fornito.
    Se il download dell'audio viene effettuato lo si trova nel percorso scritto nel codice (le cartelle della
    directory se non sono presenti vengono create automaticamente).
    Il file scaricato prende il nome del titolo del video su YouTube"""

    try:
        yt = YouTube(self.URL)
        audio = yt.streams.filter(only_audio=True).first()
        path = "C:/Users/39392/Desktop/progettoWebAnalytics/audio_mp3"
        out_file = audio.download(output_path=path)
        base, ext = os.path.splitext(out_file)
        new_file = base + '.mp3'
        try:
            os.rename(out_file, new_file)
            return "Il video su Youtbe '" + yt.title + "' è stato scaricato correttamente in formato .mp3"
        except FileExistsError:
            return f"Attention! Il file {yt.title} esiste già nel path selezionato: sono presenti più file uguali!"
    except Exception as e:
        return "Si è verificato un errore durante il download: " + str(e)

```

Attraverso il **metodo download** ho inserito il codice che permette, attraverso il richiamo dell'URL di ciascuna intervista, lo scaricamento del file audio. Ho rilanciato diverse volte il codice, ipotizzando vari errori, per capire quali possano essere i problemi che possono bloccare o influenzare il suo flusso, per questo ho inserito la gestione degli errori per indicare facilmente le possibili interruzioni di codice.

Con **yt** si crea un oggetto YouTube, che all'interno del suo parametro richiama l'URL fornito come attributo `self.url`; con **audio** si crea un oggetto in grado di filtrare il video per ottenere solo l'audio e viene selezionato il primo disponibile. Con **path** viene stabilita, attraverso una stringa richiamata successivamente, la destinazione del file audio. Se il percorso non esiste verrà creata in automatico una cartella dove si potranno trovare gli audio scaricati. Attraverso **out\_file** viene eseguito il download dell'audio che verrà salvato nel percorso specificato. Con **base** ed **ext** si divide il percorso del file in due parti: il nome del file senza estensione (`base`) e l'estensione del file (`ext`). In **new\_file** si prende la `base`, che è il file senza estensione e si aggiunge con la stringa "mp3" il formato. Con la funzione **os.rename(out\_file, new\_file)** si rinomina il file "out\_file" con il nuovo file "new\_file" che ha incluso l'estensione del formato mp3. Se il file viene scaricato correttamente c'è una riga di codice che lo indicherà attraverso il suo titolo. Se verrà mandato più volte lo stesso codice verrà scaricato ripetutamente il file audio.

```

video1 = DownloadAudioYouTube(URL="https://www.youtube.com/watch?v=S07Comn4zIk").download()
video2 = DownloadAudioYouTube(URL="https://www.youtube.com/watch?v=-szezlsJvYQ&t").download()

```

Verranno istanziati degli oggetti per la **classe DownloadAudioYouTube** e per ogni oggetto verrà inserito l'**attributo URL** e verrà chiamato il **metodo download**.

```

44      # Chiamo gli oggetti della classe DownloadAudioYouTube per ogni video da scaricare
45      print("Video su Steven Basalari:")
46      print(video1)
47      print("Video su Bianca Balti:")
48      print(video2)

```

In seguito, chiamo gli oggetti per poter procedere con lo scaricamento dei file audio in formato mp3.

### 3) Conversione da file audio (formato .mp3) a testo (formato .txt)

Considerando la lunga durata dei file mp3 estratti e la loro corposità, avendo la necessità di creare un file testo che riporti il più possibile in modo naturale ciò che viene detto durante le interviste, ho deciso di seguire le istruzioni di un video su YouTube "Transcribe Audio to Text for FREE | Whisper AI Step-by-Step Tutorial"<sup>5</sup>.

Per ottenere un testo da un audio ho usato **Whisper AI**<sup>6</sup>, un modello di apprendimento automatico finalizzato al riconoscimento vocale e alla trascrizione, creato da OpenAI. Questo modello riesce a supportare decine e decine di lingue, per questo motivo è difficile che un computer medio riesca ad avere la velocità e la potenza necessarie per poterlo utilizzare. Per avere la possibilità di usufruire del modello ho utilizzato Google Colaboraty attraverso Google Drive: questo metodo permette di scrivere codice ed eseguirlo direttamente sul browser. Per accelerare il processo si cambia l'acceleratore hardware (Runtime) che passa dall'impostazione di default CPU a T4 GPU.

Per lavorare con una base audio, ma anche con video, per poi essere trascritta in formato testo, nelle celle di Google Colaboraty ho installato i seguenti comandi:

```
▶ !pip install git+https://github.com/openai/whisper.git
```

- è un comando che installa una repository GitHub progettata in linguaggio Python che si occupa del pacchetto "**whisper**"

```
▶ !sudo apt update && sudo apt install ffmpeg
```

- è un comando che aggiorna la lista dei pacchetti del sistema operativo e installa il pacchetto "ffmpeg": strumento, molto potente, di elaborazione multimediale che viene anche usato per la conversione di file audio e video.

Leggendo il repository di Whisper<sup>7</sup> si nota che il tasso di errore per la lingua italiana nel modello è veramente basso rispetto ad altre lingue. Nostante si tenda ad indicare il modello medium per la conversione del file, testando sia il modello medium che quello large, ho notato una differenza di accuratezza del tipo di modello, in quanto quest'ultimo è nettamente superiore a quello proposto. Per questo motivo per procedere nello svolgimento del progetto userò i file in formato .txt creati dal modello large.

Per entrambi i video ho usato lo stessa procedura:

```
▶ !whisper "/content/Bianca Balti da squatter alle passerelle più prestigiose del mondo - One More Time.mp3" --model large
```

<sup>5</sup> Link video tutorial canale JenniferMarieVO: [https://www.youtube.com/watch?v=3\\_2McMS4wNM&t](https://www.youtube.com/watch?v=3_2McMS4wNM&t)

<sup>6</sup> Sito modello Whisper: <https://openai.com/index/whisper>

<sup>7</sup> Link repository GitHub Whisper: <https://github.com/openai/whisper>

- Questo tipo di comando richiama il modello large whisper e il path del file audio caricato su Google Colaboratory: creerà in automatico una serie file, in diversi formati, che alla fine del processo potranno essere scaricati, tra questi si trova la trascrizione dell'audio in formato .txt che è ciò che serve per il progetto.

I comandi e le informazioni sulla conversione degli audio in testo sono riportati nel seguente file ***Da\_Audio\_a\_Testo\_WhisperAI.ipynb***. Il file è stato ricostruito perché per poter usufruire del T4 GPU di Google Colaboratory mi sono dovuta servire di diversi account, in quanto è un servizio limitato per ogni profilo Google.

#### 4) Analisi del sentiment e delle emozioni sugli audio testo delle interviste

Nel file script ***Sentiment\_and\_Emotion\_Analysis.py*** ho creato un codice apposito per l'analisi del sentiment e delle emozioni. I dati di tale analisi saranno riportati in una cartella chiamata file\_csv attraverso dei file formato csv.

Per lavorare sull'analisi del sentiment e delle emozioni, mi sono dovuta servire della libreria **re** perchè fornisce operazioni di espressioni regolari per la lavorare con stringhe di testo e la libreria **pandas** per manipolari i dati e grazie a questa ho creato i file csv.

Ho utilizzato il modulo **Transformers** che si occupa dell'elaborazione del linguaggio naturale fornendo modelli pre-addestrati, importando la funzione **pipeline** di *Hugging Face*<sup>8</sup> che serve per semplificare l'utilizzo di modelli di trasformatori pre-addestrati per svolgere diverse attività del Natural Language Processing (NLP). Invece dal modulo **feel\_it**<sup>9</sup>(libreria installata) ho importato la classe **EmotionClassifier** che serve per il riconoscimento delle emozioni di un testo in lingua italiana. I modelli feel\_it si basano sull'architettura BERT che contiene un potente modello di trasformatori per il NLP.

```
class SentimentAndEmotionAnalysis:
    """Classe per l'analisi del sentiment e delle emozioni degli audio testo delle interviste di One More Time.
    Verrà assegnato un punteggio percentuale positivo e negativo per il sentiment, e verrà predetta un'emozione per
    ciascuna frase del corpus basata su regole lessicali. In mancanza di punteggiatura, data dal modello che ha
    convertito l'audio in testo, viene considerata una lunghezza massima di 300 caratteri per frase, quindi l'analisi
    verrà effettuata più precisamente su segmenti di testo. I risultati saranno esportati in un file CSV."""
    BASE_PATH = os.path.abspath(os.path.dirname(__file__)) # directory dello script

    def __init__(self, path_file_txt):
        """Metodo che inizializza la classe SentimentAndEmotionAnalysis, chiamato ogni volta che viene creato
        un nuovo oggetto. Inizializza l'attributo path_file_txt che indica il percorso del file testo da analizzare"""

        self.path_file_txt = path_file_txt
```

È stata creata la classe **SentimentAndEmotionAnalysis** per effettuare l'analisi del sentiment e delle emozioni delle interviste da inserire in dei file csv.

L'oggetto **BASE\_PATH** richiama la directory di dove si trova lo script e che servirà che creare al suo interno una nuova cartella in cui mettere i file csv. Nel costruttore ho inserito il percorso di ogni file txt.

---

<sup>8</sup> Sito Hugging Face - pipeline: [https://huggingface.co/docs/transformers/main\\_classes/pipelines](https://huggingface.co/docs/transformers/main_classes/pipelines)

<sup>9</sup> Sito feel\_it – Emotion Classifier: <https://towardsdatascience.com/sentiment-analysis-and-emotion-recognition-in-italian-using-bert-92f5c8fe8a2>

```

def preprocess_corpus(self):
    """Metodo che legge il file di testo del path specificato e che preprocessa il corpus del testo.
    Rimuove i newline e aggiunge un punto esclamativo alla fine dell'ultima riga per assicurarsi che le frasi
    possano essere divise secondo le formali regole del lessico. Vista la mancata punteggiatura in molte parti
    del testo, se una frase supera i 300 caratteri, viene suddivisa in segmenti con limite 300 per mantenere
    l'integrità sintattica"""

    with open(self.path_file_txt, 'r', encoding='utf-8') as file_txt:
        rows = file_txt.readlines()
        rows[-1] = rows[-1].rstrip() + '!' # Aggiunge '!' alla fine dell'ultima riga della lista rows

    corpus = "\n".join([row.replace("\n", "") for row in rows])
    pattern = r'^.*?(!|\.{3}|[.?!])'
    sentences = re.findall(pattern, corpus)

    preprocessed_segments = []
    for sentence in sentences:
        if len(sentence) <= 300:
            preprocessed_segments.append(sentence.strip())
        else:
            while len(sentence) > 300:
                last_space_idx = sentence.rfind(' ', 0, 300)
                if last_space_idx == -1:
                    last_space_idx = 300
                preprocessed_segments.append(sentence[:last_space_idx].strip())
                sentence = sentence[last_space_idx:].strip()
            preprocessed_segments.append(sentence.strip())

```

Il metodo **preprocess\_corpus** gestisce il percorso specificato per il testo ed esegue il preprocessing del corpus. Rimuove le newline e aggiunge un punto esclamativo alla fine di ogni testo per consentire una corretta divisione in frasi. Le frasi vengono inizialmente suddivise secondo le regole del lessico utilizzando la funzione **findall** dalla libreria **re**, considerando la mancanza di punteggiatura in molte parti del testo, probabilmente a causa della conversione da audio a testo. Questa suddivisione include un limite di 300 caratteri per creare segmenti di testo separati, che possono poi essere utilizzati per l'analisi del sentiment e delle emozioni. Ho anche aggiunto il metodo **strip()** per ogni segmento di testo per rimuovere spazi vuoti e prevenire errori nel codice durante l'analisi.

```

def sentiment_and_emotion_analysis(self):
    """Metodo che permette l'analisi del sentiment e delle emozioni di ciascun segmento di testo. Sono stati usati
    modelli pre-addestrati che fanno predizione del sentiment (positiva e negativa, indicata in percentuale) e delle
    emozioni (indicate come: joy, sadness, fear, anger).
    I dati per ciascun segmento di testo verrà salvato in un file csv all'interno di una cartella specifica"""

    # Caricamento del modello per l'analisi del sentiment del testo in lingua italiana
    sentiment_classifier = pipeline("text-classification", model='MilaNLProc/feel-it-italian-sentiment', top_k=2)

    # Caricamento della funzione per l'analisi delle emozioni del testo in lingua italiana
    emotion_classifier = EmotionClassifier()

    preprocessed_segments = self.preprocess_corpus()

```

Il metodo **sentiment\_and\_emotion\_analysis** serve per eseguire l'analisi del sentiment e delle emozioni delle frasi delle parti di testo che rispondono ai requisiti del metodo precedente. Per effettuare un'analisi su ogni frase sono usati modelli pre-addestrati per l'italiano. Per il sentiment viene creato l'oggetto **classifier** in cui viene passata la funzione **pipeline** che ha come parametro il modello pre-addestrato '**MilaNLProc/feel-it-italian-sentiment**'<sup>10</sup>. Questo modello restituisce la percentuale positiva e negativa del testo analizzato.

---

<sup>10</sup> Spiegazione modello MilaNLProc; <https://huggingface.co/MilaNLProc/feel-it-italian-sentiment>

Per le emozioni viene creato un oggetto **emotion\_classifier** che passa la funzione **EmotionClassifier()** che restituisce un tipo di emozione per ogni frase. Le emozioni sono : joy, fear, anger, sadness.

```
results = []
for segment in preprocessed_segments:
    try:
        sentiment_predictions = sentiment_classifier(segment)[0]
        pos_score = next((item['score'] for item in sentiment_predictions if item['label'] == 'positive'), 0)
        neg_score = next((item['score'] for item in sentiment_predictions if item['label'] == 'negative'), 0)
        predicted_emotion = emotion_classifier.predict([segment])

        results.append({
            'Frasi': segment,
            'Sentiment Positivo (%)': pos_score,
            'Sentiment Negativo (%)': neg_score,
            'Emozione Predetta': predicted_emotion
        })
    except Exception as e:
        print(f"Errore durante l'analisi della frase: {e}")

df = pd.DataFrame(results)
```

In questo frammento di codice viene eseguita un'analisi per ciascuna frase di testo. Richiamando modelli pre-addestrati, vengono effettuate predizioni per il sentiment e le emozioni. I risultati di queste analisi vengono poi inseriti in un DataFrame utilizzando la libreria Pandas. Per il sentiment viene riportato un valore che indica quanto la frase è positiva e negativa attraverso un punteggio che va da 0 a 1; per le emozioni viene indicata l'emozione predetta che secondo il modello rappresenta la frase.

```
output_folder = os.path.join(self.BASE_PATH, 'file_csv')
os.makedirs(output_folder, exist_ok=True)
output_csv_path = os.path.join(output_folder, os.path.basename(self.path_file_txt)[0] + ".csv")
df.to_csv(output_csv_path, index=False)
```

L'oggetto **output\_folder** indica il percorso per la creazione della cartella destinata ai file CSV, che viene creata utilizzando la funzione **os.makedirs**. Invece, **output\_csv\_path** indica il percorso specifico per ciascun file CSV che viene creato utilizzando **df.to\_csv** per esportare il DataFrame in un file CSV all'interno della cartella indicata.

```
# Creazione istanze per la classe SentimentAndEmotionAnalysis che richiama l'attributo path_file_txt del costruttore
StevenBasalari = SentimentAndEmotionAnalysis(path_file_txt=r"C:\Users\39392\Desktop\progettoWebAnalytics\file_txt\BiancaBalti_model_large.txt")
BiancaBalti = SentimentAndEmotionAnalysis(path_file_txt=r"C:\Users\39392\Desktop\progettoWebAnalytics\file_txt\StevenBasalari_model_large.txt")

if __name__ == "__main__":
    # Chiamata classe con il metodo sentiment_and_emotion_analysis()
    print("Analisi per l'intervista di Steven Basalari: ", end="")
    print(StevenBasalari.sentiment_and_emotion_analysis())
    print("\n")
    print("Analisi per l'intervista di Bianca Balti: ", end="")
    print(BiancaBalti.sentiment_and_emotion_analysis())
```

**StevenBasalari** e **BiancaBalti** rappresentano le istanze della classe in cui viene passato l'attributo specifico per ciascuna intervista. Successivamente viene chiamato il metodo che effettua l'analisi del sentiment e delle emozioni che verranno riportate in file csv.

## 5) Visualizzazioni grafiche: analisi di sentiment ed emozioni sull' audio testo delle interviste

**Grafici\_SentimentEmotion.py** è lo script attraverso cui sono stati creati i grafici che permettono in modo semplice la visualizzazione dell'occorrenza del sentiment positivo e negativo, e delle emozioni per le frasi dei testi delle interviste attraverso la costruzione della classe **GraficiSentimentEmotion**.

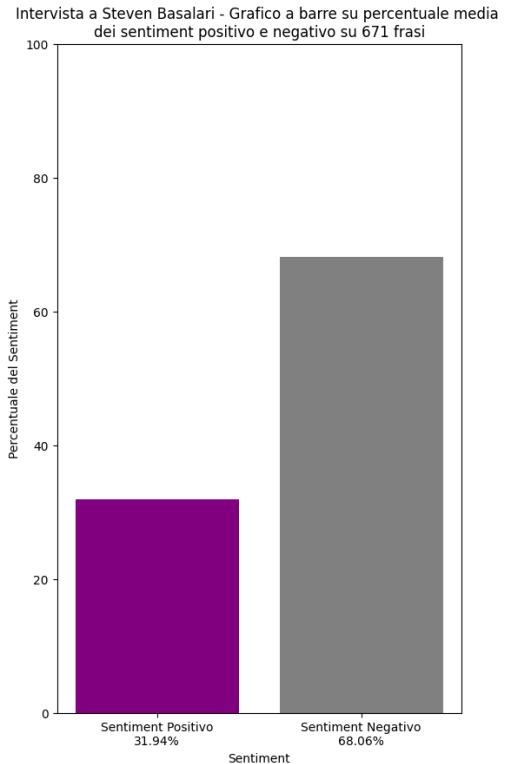
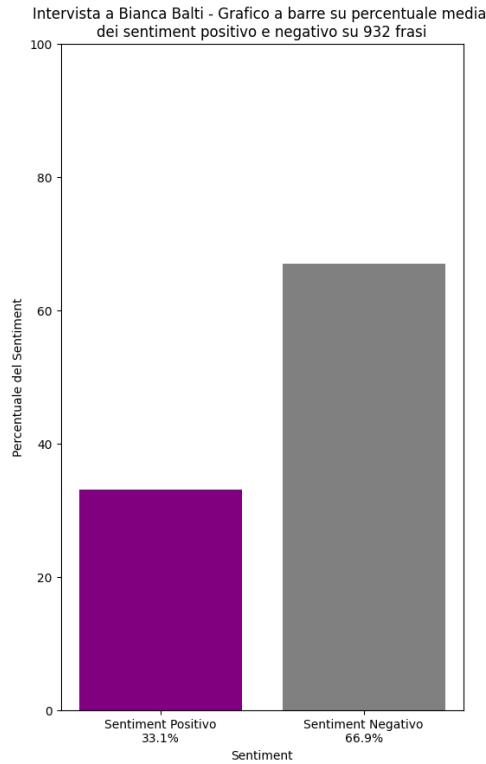
```
class GraficiSentimentEmotion:  
    """Classe creata per gestire la creazione di diversi grafici basati sul dataset di frasi che compongono il testo audio delle interviste, in cui viene espressa la percentuale positiva e negativa del sentiment, e l'emozione predetta per ciascuna frase."""  
  
    def __init__(self, path, nome):  
        """Il costruttore della classe viene inizializzato con il percorso del file CSV che contiene il dataset con le informazioni sul sentiment e le emozioni di ciascuna frase che compone il testo dell'intervista, e con il nome dell'intervistato. Viene creato anche l'attributo df che è utile per richiamare la lettura dei file CSV tramite il percorso specificato."""  
  
        self.path = path  
        self.nome = nome  
        try:  
            self.df = pd.read_csv(self.path)  
        except FileNotFoundError:  
            print(f"Il file csv non è trovato")  
            self.df = None
```

Nel metodo costruttore della classe ci sono due argomenti: il percorso del file CSV (**path**) e il nome dell'intervistato(**nome**). Si inizializzano gli attributi e si tenta di leggere il file CSV mediante pandas se il file esiste, in caso contrario viene impostato a none.

```
def graficoBarreSentiment(self):  
    """Metodo che analizza il sentiment medio delle frasi nell'intervista. Attraverso un grafico a barre viene mostrata la media percentuale del sentiment negativo e positivo delle frasi. Viene espresso anche il totale delle frasi per intervista."""  
  
    frasi = self.df['Frase']  
    sentiment_pos = self.df['Sentiment Positivo (%)']  
    sentiment_neg = self.df['Sentiment Negativo (%)']  
  
    print("Totale frasi:", len(frasi))  
    tot_sentiment_pos = round(sentiment_pos.mean() * 100, 2)  
    tot_sentiment_neg = round(sentiment_neg.mean() * 100, 2)  
  
    print("Percentuale approssimativa Sentiment Positivo su tutte le frasi:", tot_sentiment_pos, "%")  
    print("Percentuale approssimativa Sentiment Negativo su tutte le frasi:", tot_sentiment_neg, "%")  
  
    labels = [f'Sentiment Positivo\n{tot_sentiment_pos}%', f'Sentiment Negativo\n{tot_sentiment_neg}%']  
    values = [tot_sentiment_pos, tot_sentiment_neg]  
    plt.figure(figsize=(6, 10))  
    plt.bar(labels, values, color=['purple', 'grey'])  
    plt.xlabel('Sentiment')  
    plt.ylabel('Percentuale del Sentiment')  
    plt.title(f"Intervista a {self.nome} - Grafico a barre su percentuale media \n dei sentiment positivo e negativo su {len(frasi)} frasi")  
    plt.ylim(0, 100)  
    plt.show()  
    plt.savefig(os.path.join('Grafici_Immagini', f'{self.nome}_grafico_barre_sentiment.png'), bbox_inches='tight')
```

Attraverso il metodo **graficoBarreSentiment** si crea un grafico a barre che mostra la percentuale media di sentiment positivo e negativo per tutte le frasi nell'intervista e viene anche mostrato il numero totale di frasi per intervista. Il grafico viene salvato nella cartella

**Grafici\_Immagini.** Per il sentiment negativo si ha colore grigio e per quello positivo il colore viola.



1. *Grafici a barre che indicano la percentuale media dei sentiment positivi e negativi sulle frasi delle interviste di Bianca Balti e Steven Basalari*

Dal grafico si evince che, a livello di percentuali medie sui sentiment, i valori per entrambe le interviste siano simili e che il sentiment negativo prevalga di gran lunga su quello positivo.

```
def graficoSentimentSerieTemporale(self):
    """Metodo che esamina le variazioni del sentiment positivo e negativo nel tempo, calcolando la media dei due tipi di sentiment ogni 30 frasi, a parte l'ultimo gruppo che è costituito da meno frasi. Viene rappresentato graficamente attraverso barre accostate, mostrando la percentuale di negatività e positività presente durante lo svolgimento dell'intervista."""

    frasi = self.df['Frasi']
    sentiment_pos = self.df['Sentiment Positivo (%)']
    sentiment_neg = self.df['Sentiment Negativo (%)']
    dimensione_gruppo = 30
    num_gruppi = len(frasi) // dimensione_gruppo + 1
    print("Numero gruppi per rappresentazione grafico a barre accostate:", num_gruppi)

    media_sentiment_pos = []
    media_sentiment_neg = []
    for i in range(num_gruppi):
        idx_inizio = i * dimensione_gruppo
        idx_fine = min((i + 1) * dimensione_gruppo,
                      len(frasi))
        media_pos = sentiment_pos[idx_inizio:idx_fine].mean()
        media_neg = sentiment_neg[idx_inizio:idx_fine].mean()
        media_sentiment_pos.append(media_pos)
        media_sentiment_neg.append(media_neg)
```

```

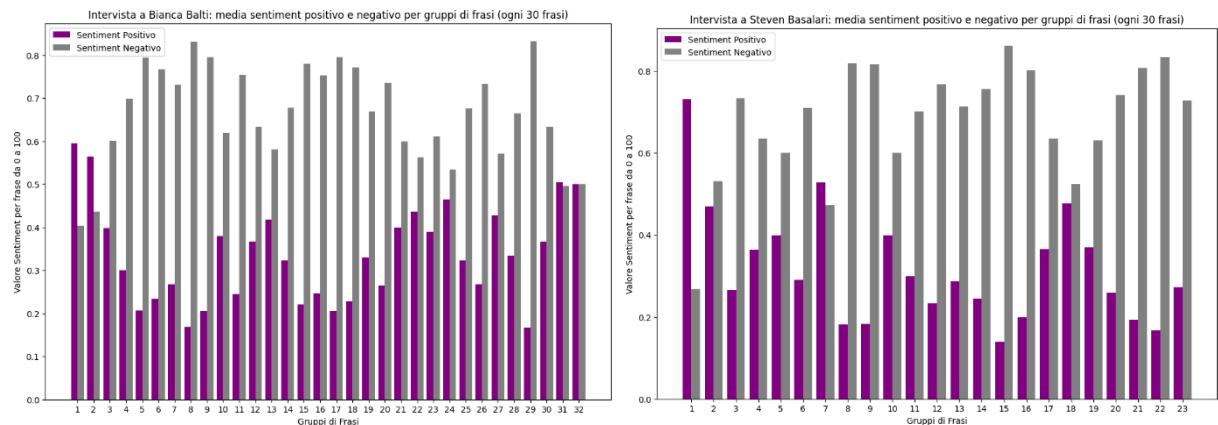
plt.figure(figsize=(12, 8))
x = range(num_gruppi)
larghezza = 0.4
plt.bar(x, media_sentiment_pos, width=larghezza, label='Sentiment Positivo', color='purple', align='center')
plt.bar([i + larghezza for i in x], media_sentiment_neg, width=larghezza, label='Sentiment Negativo',
        color='grey',
        align='center')

plt.xlabel('Gruppi di Frasi')
plt.ylabel('Valore Sentiment per frase da 0 a 100')
plt.xticks([i + larghezza / 2 for i in x], [f'{i + 1}' for i in x])
plt.title(
    f"Intervista a {self.nome}: media sentiment positivo e negativo"
    f" per gruppi di frasi (ogni {dimensione_gruppo} frasi)")
plt.legend()

plt.show()
plt.savefig(os.path.join('Grafici_Immagini', f'{self.nome}_grafico_serie_temporale_sentiment.png'),
            bbox_inches='tight')

```

Il metodo **graficoSentimentSerieTemporale**, rappresentato in questi frammenti di codice, crea un grafico che mostra le variazioni del sentiment positivo e negativo nel tempo, calcolando la media dei due tipi di sentiment ogni 30 frasi (tranne l'ultimo gruppo che al massimo è costituito da meno di 30 frasi). Il grafico utilizza barre accostate per rappresentare la percentuale positiva e negativa delle varie frasi durante l'intervista.



## 2. Grafici a barre accostate che mostrano la variazione del sentiment negativo e positivo per ogni intervista.

Dai grafici si nota come, a parte l'inizio per entrambe le interviste, il sentiment negativo prevalga di gran lunga su quello positivo. Solo alla fine dell'intervista di Bianca Balta c'è una sorta di uguaglianza tra il sentiment negativo e quello positivo.

```

def graficoTortaEmozioni(self):
    """Metodo che analizza e visualizza, tramite un grafico a torta, la presenza media di un certa emozione nelle frasi dell'intervista. Il valore medio è espresso in percentuale"""

    df = self.df
    conteggio_emozioni = {}
    totale_emozioni = 0
    for emozioni_str in self.df['Emozione Predetta']:
        emozioni = eval(emozioni_str)
        totale_emozioni += len(emozioni)
        for emozione in emozioni:
            if emozione in conteggio_emozioni:
                conteggio_emozioni[emozione] += 1
            else:
                conteggio_emozioni[emozione] = 1

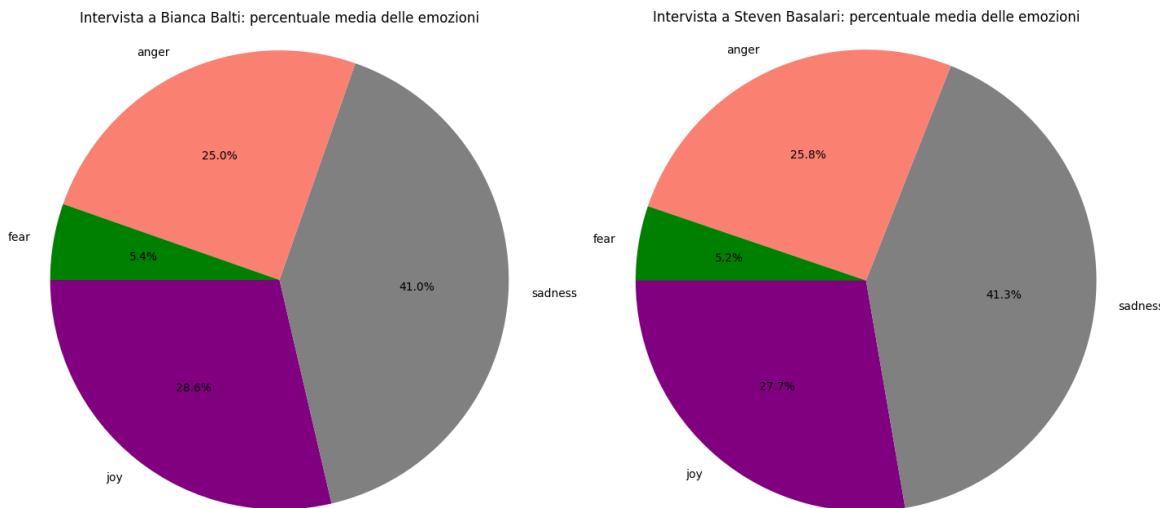
    percentuali_medie = {emozione: (conteggio / totale_emozioni * 100) for emozione, conteggio in conteggio_emozioni.items()}

    emozioni = list(percentuali_medie.keys())
    percentuali = list(percentuali_medie.values())
    colori_emozioni = {'joy': 'purple', 'sadness': 'grey', 'fear': 'green', 'anger': 'salmon'}
    colors = [colori_emozioni[emozione] for emozione in emozioni]

    plt.figure(figsize=(8, 8))
    plt.pie(percentuali, labels=emozioni, autopct='%.1f%%', startangle=180, colors=colors)
    plt.title(f"Intervista a {self.nome}: percentuale media delle emozioni")
    plt.axis('equal')

```

Nel metodo **graficoTortaEmozioni** viene creato un grafico a torta che mostra la percentuale media di ogni emozione (gioia, tristezza, paura, rabbia) presente nelle frasi dell'intervista. Per ogni frase c'è solo un tipo di emozione predetta.



### 3. Grafici a torta che rappresentano la percentuale di emozioni predette su tutte le frasi per ogni testo delle interviste

Dai grafici si può notare una similitudine nelle percentuali delle emozioni delle interviste. L'emozione predominante è la tristezza, che raggiunge un valore superiore al 40%, seguita dalla gioia, che per entrambe le interviste si attesta intorno al 28%, e dalla rabbia, che si aggira intorno al 25%. Infine, c'è la paura, che per entrambe supera di poco il 5%.

```

def graficoSerieTemporaleEmozioni(self):
    """Metodo che analizza dal punto vista temporale la presenta ci un certo quantitativo di emozioni ogni 30 frasi,
    a parte l'ultimo gruppo composto da un numero minore. Attraverso dei marcatori (uno diverso per ogni emozione)
    è possibile vedere l'andamento di ciascuna emozione durante l'intervista."""

    df = self.df
    frasi = self.df['Frasi']

    dimensione_gruppo = 30
    num_gruppi = len(frasi) // dimensione_gruppo + 1
    print("Numero gruppi per rappresentazione grafica per serie temporale sulle emozioni:", num_gruppi)

    occorrenze_emozioni = {emozione: [0] * num_gruppi for emozione in ['joy', 'sadness', 'fear', 'anger']}
    for i in range(num_gruppi):
        idx_inizio = i * dimensione_gruppo
        idx_fine = min((i + 1) * dimensione_gruppo, len(df))
        gruppo_fras = df['Emozione Predetta'][idx_inizio:idx_fine]

        for emozioni_str in gruppo_fras:
            emozioni = eval(emozioni_str)
            for emozione in emozioni:
                occorrenze_emozioni[emozione][i] += 1

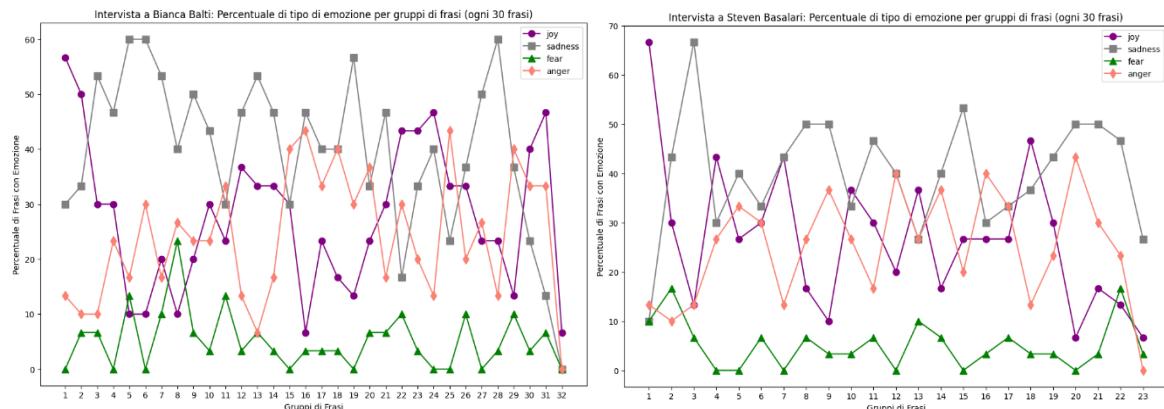
    for emozione in occorrenze_emozioni:
        for i in range(num_gruppi):
            occorrenze_emozioni[emozione][i] = (occorrenze_emozioni[emozione][i] / dimensione_gruppo) * 100

    for emozione, occorrenze in occorrenze_emozioni.items():
        x = range(1, num_gruppi + 1)
        plt.plot(x, occorrenze, label=emozione, color=colori[emozione], marker=marker[emozione], markersize=8)

    plt.xlabel('Gruppi di Frasi')
    plt.ylabel('Percentuale di Frasi con Emozione')
    plt.xticks([i for i in range(1, num_gruppi + 1)], [f'{i}' for i in range(1, num_gruppi + 1)])
    plt.title(f"Intervista a {self.nome}: Percentuale di tipo di emozione per gruppi di frasi (ogni {dimensione_gruppo} frasi)")
    plt.legend()
    plt.show()
    plt.savefig(os.path.join('Grafici_Immagini', f'{self.nome}_grafico_serie_temporale_emozioni.png'),
               bbox_inches='tight')

```

Nel metodo **graficoSerieTemporaleEmozioni** si crea un grafico che mostra le variazioni delle emozioni (gioia, tristezza, paura, rabbia) nel tempo, calcolando la percentuale di frasi che contengono ciascuna emozione ogni 30 frasi (tranne l'ultimo gruppo che potrebbe avere meno di 30 frasi). Il grafico utilizza marcatori diversi per rappresentare ogni emozione.



4. *Grafici che indicano le variazioni delle emozioni durante le interviste (quanto è frequente un'emozione a livello percentuale ogni 30 frasi)*

Dai grafici si nota che l'intervista di Bianca Balti presenta emozioni più frastagliate rispetto a quella di Steven Basalari, che sembra essere più coerente nelle emozioni. Le emozioni maggiormente presenti sono la tristezza e la gioia in entrambi i grafici, al contrario della paura.

```
# Si istanziano oggetti per la classe GraficiSentimentEmotion che hanno come attributo il percorso del dataset di
# ciascuna intervista e il nome della persona intervistata
StevenBasalari = GraficiSentimentEmotion(
    path=r"C:\Users\39392\Desktop\progettoWebAnalytics\file_csv\StevenBasalari_model_large.csv",
    nome="Steven Basalari")
BiancaBalti = GraficiSentimentEmotion(
    path=r"C:\Users\39392\Desktop\progettoWebAnalytics\file_csv\BiancaBalti_model_large.csv",
    nome="Bianca Balti")

if __name__ == "__main__":
    # Richiamo oggetti della classe e successivamente i vari metodi
    print("Steven Basalari:")
    StevenBasalari.graficoBarreSentiment()
    StevenBasalari.graficoSentimentSerieTemporale()
    StevenBasalari.graficoTortaEmozioni()
    StevenBasalari.graficoSerieTemporaleEmozioni()
    print("Bianca Balti:")
    BiancaBalti.graficoBarreSentiment()
    BiancaBalti.graficoSentimentSerieTemporale()
    BiancaBalti.graficoTortaEmozioni()
    BiancaBalti.graficoSerieTemporaleEmozioni()
```

In questa parte di codice vengono istanziati gli oggetti per la classe che hanno come attributo il percorso del file CSV con i rispettivi dati dell'analisi del sentimento e delle emozioni, e il nome dell'intervistato. Successivamente per ogni intervista viene richiamata l'istanza con i vari metodi della classe.

## 6) Analisi dei topic rilevanti sulle frasi dei testi delle interviste

**TopicModelling.py** è lo script dove viene implementato un codice che si occupa di effettuare un'analisi dei topic sui testi delle interviste. Inizialmente queste vengono preprocessate e per ognuna di esse viene creato un modello LDA (Latent Dirichlet Allocation) per estrarre i topic principali dal testo. I risultati vengono analizzati attraverso grafici, word clouds e interfacce interattive. Infine, viene eseguita un'analisi comparativa dei topic dominanti tra due interviste utilizzando il test del Chi-quadro e per ogni intervista viene determinato il livello di coerenza dei topic estratti.

```
class TopicModel:
    """ La classe TopicModel è progettata per analizzare i topic all'interno di file csv che contengono la trascrizione
    audio convertita in testo per frasi. Effettua il preprocessing testuale in lingua italiana utilizzando la libreria
    spacy e gestisce l'estrazione dei topic utilizzando il modello Latent Dirichlet Allocation (LDA) di gensim. I
    risultati verranno salvati in una cartella che conterrà le analisi rappresentative sotto forma di wordclouds e HTML """
    # laruiu *
    def __init__(self, file_path, output_folder):
        """ Nel costruttore vengono inizializzati gli attributi di istanza della classe che gestiscono la lettura del file
        CSV, l'estrazione del titolo utilizzato per i grafici e l'inizializzazione del modello LDA e dei topic,
        inizialmente impostati a None per essere utilizzati successivamente durante l'analisi. Si inizializza
        l'insieme predefinito di stopwords in lingua italiana da spacy e l'aggiunta di altre stopwords per avere maggiore
        accuratezza nell'analisi. Infine si inizializza la cartella dove verranno importati i risultati dei topic in formato
        di immagini e html. """

```

```

    self.df = pd.read_csv(file_path)
    self.csv_title = os.path.splitext(os.path.basename(file_path))[0]
    self.lda_model = None
    self.topics = None
    self.nlp = spacy.load('it_core_news_sm')
    self.stop_words_italian = self.nlp.Defaults.stop_words
    self.additional_stopwords = [
        'che', 'per', 'con', 'come', 'quando', 'sono', 'abbiamo', 'avete', 'hanno', 'sia', 'siamo', 'siete', 'essere',
        'avere', 'fare', 'dire', 'detto', 'fatto', 'del', 'lha', 'dei', 'della', 'delle', 'nell', 'nella', 'nelle',
        'nello', 'negli', 'come', 'laltro', 'quando', 'alla', 'allo', 'alle', 'agli', 'senza', 'sotto', 'sopra',
        'tipo', 'los', 'lodo', 'boh', 'sai', 'cioè', 'parlarepadre', 'cavolo', 'lo', 'milo', 'cerare', 'bello', 'mettere',
        'dare', 'dici', 'mille', 'sacco', 'vero', 'cosa', 'lo', 'buttare', 'sentire', 'andare', 'chiedere', 'punto',
        'giusto', 'super', 'perchè', 'ventanni', 'maturo', 'ventenne', 'finché', 'grazia', 'domanda', 'stare', 'volere',
        'potere', 'priorità', 'mese', 'labbiamo', 'dovere', 'possibilità', 'base', 'giro', 'effetto', 'venire', 'minuto',
        'periodo', 'mese', 'settimana', 'dieci', 'cinque', 'settimana', 'lha', 'laltra', 'problema', 'maniera', 'roba',
        'discorso', 'nome', 'struttura', 'bisogno', 'cacao', 'aprile']
    self.output_folder = output_folder

```

In questa parte di codice viene definita la classe **"TopicModel"**, progettata per analizzare i topic all'interno di file CSV contenenti le frasi trascritte in testo degli audio delle interviste.

Il **costruttore della classe** richiamato per ogni istanza definita successivamente, inizializza attributi come il DataFrame df che contiene i dati del file CSV, il titolo del file CSV, il modello LDA e i topic (inizialmente impostati a None), il modello di linguaggio SpaCy per l'italiano (self.nlp), l'insieme di stopwords predefinite in italiano, un elenco aggiuntivo di stopwords personalizzate e la cartella di output per salvare i risultati.

Le stopwords personalizzate sono state aggiunte man mano che il codice veniva provato, per migliorare i risultati dell'analisi dei topic.

```

def preprocess_texts(self):
    """Metodo che esegue il preprocessing dei testi contenuti nel DataFrame self.df['Frase'] dove ogni frase viene
    trasformata in una lista di token significativi e il risultato viene memorizzato nella nuova colonna chiamata
    Tokens"""

    self.df['Tokens'] = self.df['Frase'].apply(self.tokenize_text)

```

Il metodo **preprocess\_texts** esegue il preprocessing dei testi contenuti nella colonna "Frase" del DataFrame **self.df**. Ogni frase del testo dell'intervista viene trasformata in una lista di token significativi utilizzando il metodo **tokenize\_text** e il risultato viene memorizzato in una nuova colonna chiamata "**Tokens**".

```

def tokenize_text(self, text):
    """Metodo che esegue la tokenizzazione e il preprocessing utilizzando spacy. Viene eliminata la punteggiatura e
    il testo viene convertito in minuscolo. Viene creato un oggetto doc per il testo tokenizzato e c'è l'iterazione
    su ciascun token applicando determinati criteri: i token non devono essere segni di punteggiatura o numeri,
    devono avere almeno 2 caratteri e non devono essere pronomi, articoli, verbi, preposizioni, avverbi o aggettivi.
    I token validi vengono sottoposti alla lemmatizzazione per ottenere il lemma e verificare che non siano stopwords.
    Infine, i token preprocessati vengono restituiti come risultato del metodo attraverso una lista"""

    text = re.sub(pattern=r'[^w\s]', repl='', text.lower())
    doc = self.nlp(text)
    tokens = []
    for token in doc:
        if not token.is_punct and not token.like_num and len(token.text) > 2 and token.pos_ not in ['PRON', 'DET',
                                                                                                     'VERB', 'ADP',
                                                                                                     'ADV', 'ADJ']:
            lemma = token.lemma_
            if lemma not in self.stop_words_italian and lemma not in self.additional_stopwords:
                tokens.append(lemma)
    return tokens

```

Il metodo **tokenize\_text** esegue la tokenizzazione e il preprocessing del testo utilizzando la libreria SpaCy. Inizialmente viene rimossa la punteggiatura e il testo viene convertito in minuscolo attraverso la funzione **sub** della libreria **re**. Poi viene creato un oggetto **doc** per il testo tokenizzato attraverso la funzione **nlp** della libreria **SpaCy** e inizializzata una lista vuota per i token che verranno poi preprocessati. Attraverso un'iterazione con il ciclo for su ciascun token su doc, vengono applicati dei criteri per selezionare token validi per effettuare l'analisi dei topic. Oltre all'esclusione della punteggiatura, non vengono inseriti i numeri e i token devono avere più di due caratteri, e non devono essere categorizzati come pronomi, avverbi, articoli e verbi. I token restanti vengono lemmatizzati e se non appartengono alle stopwords predefinite e personalizzate, vengono aggiunti alla lista tokens, che alla fine del metodo viene restituita.

```
def lda(self):
    """Metodo che esegue l'estrazione dei topic utilizzando il modello LDA. Vengono impostati i seed per fare in modo
    che escano sempre gli stessi risultati. I token preprocessati vengono convertiti in un Dictionary e poi in un
    bag-of-words corpus. Successivamente, viene creato un modello LDA con parametri specifici come il numero di
    topic (5) e il numero di iterazioni (3000). Le informazioni sui topic vengono memorizzate in un dizionario e per
    ciascun topic vengono stampate le parole chiave"""

    random.seed(10)
    np.random.seed(10)
    tokens = self.df['Tokens'].tolist()
    dict = corpora.Dictionary(tokens)
    bow_corpus = [dict.doc2bow(t) for t in tokens]
    self.lda_model = gensim.models.ldamodel.LdaModel(
        corpus=bow_corpus,
        id2word=dict,
        num_topics=5,
        random_state=10,
        update_every=1,
        chunksize=100,
        passes=200,
        alpha='auto',
        iterations=3000,
        per_word_topics=True
    )
    self.topics = {idx: topic for idx, topic in self.lda_model.print_topics(num_words=10)}
    print("Topic per intervista: ")
    print(self.topics)
```

In questa parte di codice viene definito il metodo **Ida** che si occupa dell'estrazione di topic utilizzando il modello **LDA (Latent Dirichlet Allocation)**. Dopo aver provato per diverse volte il codice per interno, sono stati impostati i seed per garantire la riproducibilità dei risultati generati dal modello LDA. I token preprocessati vengono convertiti in un dizionario e poi in un bag-of-words corpus utilizzando la classe **Dictionary** di Gensim. Dopo viene istanziato un oggetto **Ida\_model** specificando dei parametri come il corpus creato precedentemente, il numero di topic, il seed per la generazione di numeri casuali, il numero di passaggi sull'intero corpus durante l'addestramento e il numero massimo di iterazioni sull'intero corpus. Successivamente le informazioni sui topic vengono memorizzate in un dizionario **topics**, dove ciascuna chiave rappresenta l'ID del topic e il valore è una lista di parole chiave associate a quel topic. Infine, vengono visualizzati i topic per l'intervista specifica.

```

def topic_plot(self):
    """Metodo che crea le word cloud relative ai topic estratti dal modello LDA. Per ogni topic viene creata un'immagine basata sulla probabilità delle parole chiave (più è grande la parola più quella parola ha probabilità maggiore). Le immagini vengono salvate in un file png all'interno della cartella dei risultati dell'analisi sui topic"""

    name_plot = os.path.join(self.output_folder, f'{self.csv_title}_Grafico_Topic.png')
    plt.figure(figsize=(30, 30))
    sns.set_theme(style="darkgrid")
    for i in range(5):
        df = pd.DataFrame(self lda_model.show_topic(i), columns=['Token', 'Probabilità']).set_index('Token')
        df = df.sort_values('Probabilità')
        plt.subplot(*args: 5, 2, i + 1)
        plt.title('Topic ' + str(i))
        sns.barplot(x='Probabilità', y=df.index, data=df, palette='Greens_d')
        plt.xlabel('Probabilità')
    plt.suptitle(t=f'Modello di Topic - Frasi da {self.csv_title}', fontsize=50)
    plt.savefig(name_plot)
    plt.show()
    print("Il topic plot è disponibile!")

def topic_plot(self):
    """Metodo che crea le word cloud relative ai topic estratti dal modello LDA. Per ogni topic viene creata un'immagine basata sulla probabilità delle parole chiave (più è grande la parola più quella parola ha probabilità maggiore). Le immagini vengono salvate in un file png all'interno della cartella dei risultati dell'analisi sui topic"""

    name_plot = os.path.join(self.output_folder, f'{self.csv_title}_Grafico_Topic.png')
    plt.figure(figsize=(30, 30))
    sns.set_theme(style="darkgrid")
    for i in range(5):
        df = pd.DataFrame(self lda_model.show_topic(i), columns=['Token', 'Probabilità']).set_index('Token')
        df = df.sort_values('Probabilità')
        plt.subplot(*args: 5, 2, i + 1)
        plt.title('Topic ' + str(i))
        sns.barplot(x='Probabilità', y=df.index, data=df, palette='Greens_d')
        plt.xlabel('Probabilità')
    plt.suptitle(t=f'Modello di Topic - Frasi da {self.csv_title}', fontsize=50)
    plt.savefig(name_plot)
    plt.show()
    print("Il topic plot è disponibile!")

def word_cloud(self):
    """Metodo che calcola la coerenza dei topic estratti usando il modello LDA attraverso un punteggio che va da 0 a 1. L'interpretazione della coerenza risulta soggettiva e dipende dal contesto specifico in cui viene effettuata l'analisi di topic"""

    name_cloud = os.path.join(self.output_folder, f'{self.csv_title}_Word_Cloud.png')
    plt.figure(figsize=(30, 30))
    for i in range(5):
        df = pd.DataFrame(self lda_model.show_topic(i), columns=['Token', 'Probabilità']).set_index('Token')
        df = df.sort_values(by='Probabilità', ascending=False)
        wordcloud = WordCloud(width=800,
                              height=300,
                              background_color='white').generate_from_frequencies(dict(df['Probabilità']))
        plt.subplot(*args: 5, 2, i + 1)
        plt.title('Topic ' + str(i))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
    plt.suptitle(t='Word Clouds - Topics', fontsize=50)
    plt.savefig(name_cloud)
    plt.show()
    print("Le word cloud per ogni topic sono disponibili!")

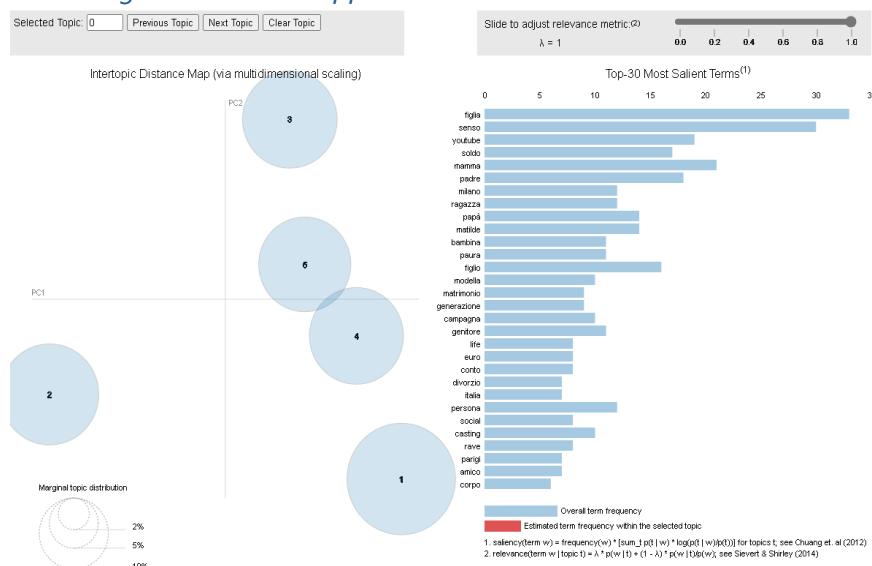
```

In questi frammenti di codice vengono definiti tre metodi utilizzati dal modello LDA per l'analisi

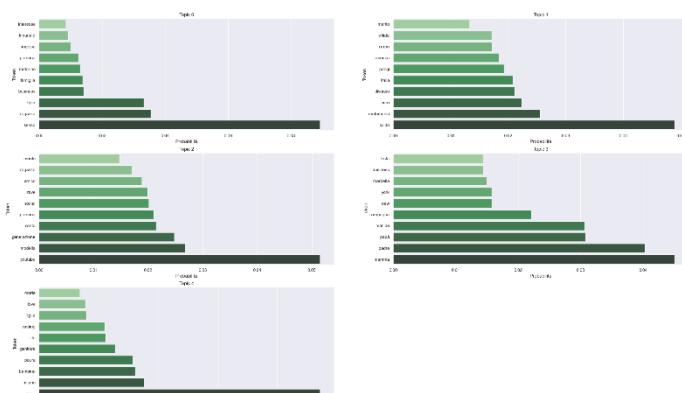
dei topic per visualizzare i risultati di ogni intervista.

- Nel metodo **save\_visualization** si genera una visualizzazione interattiva dei topic utilizzando la libreria **pyLDAvis**, dove viene creata un'immagine HTML che mostra la distribuzione delle parole chiave per ciascun topic ordinato per probabilità. L'HTML risultante viene salvato nella cartella dei risultati dell'analisi dei topic.
- Nel metodo **topic\_plot** si creano i grafici a barre per i topic estratti dal modello LDA. Viene utilizzata la libreria **seaborn** per la creazione dei grafici, dove vengono mostrate le probabilità delle parole chiave per ciascun topic. I grafici vengono salvati come file PNG nella cartella dei risultati dell'analisi dei topic.
- Nel metodo **word\_cloud** vengono generate delle word cloud per i topic estratti dal modello LDA. Ogni word cloud rappresenta le parole chiave per un determinato topic, dimensionate in base alla loro probabilità. Vengono salvate con la stessa modalità del metodo precedente.

## 5. Risultati immagini dei metodi appena elencati dell'intervista a Bianca Balti



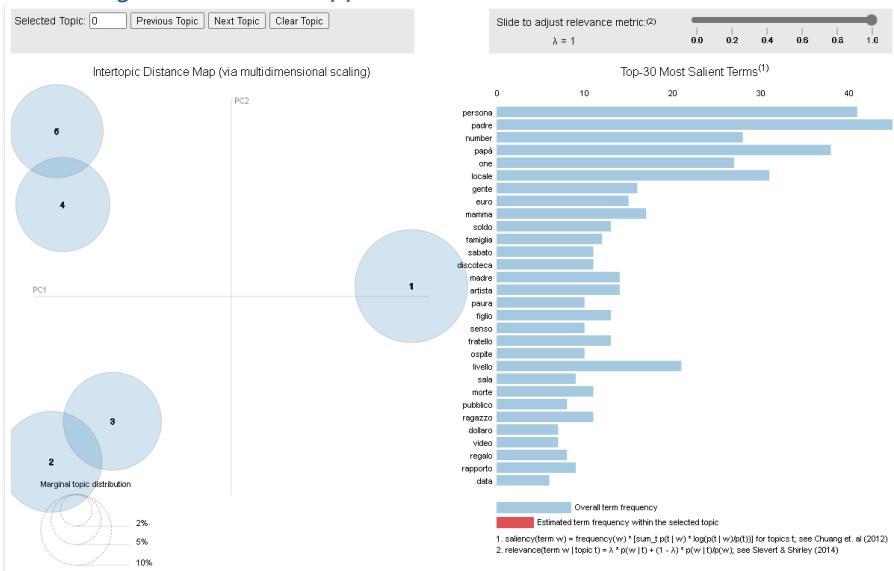
Modello di Topic - Frasi da BiancaBalti\_model\_large



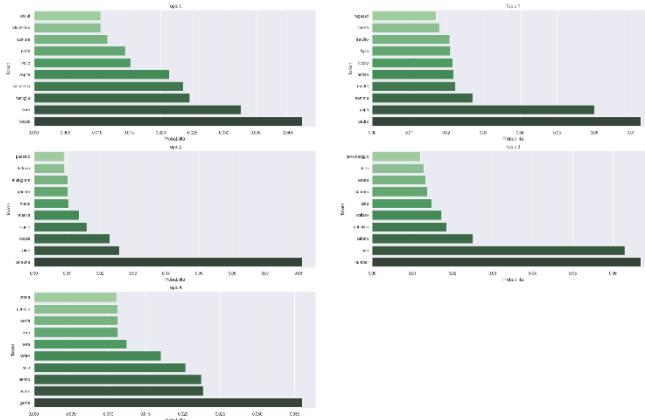
Word Clouds - Topics



## 6. Risultati immagini dei metodi appena elencati dell'intervista a Steven Basalari



Modello di Topic - Frasi da StevenBasalari\_model\_large



Word Clouds - Topics



Per ogni intervista sono stati selezionati cinque topic principali basati sulle frasi degli audio testi delle interviste. L'obiettivo di questa analisi è comprendere la distribuzione dei topic, i temi ricorrenti nei discorsi degli intervistati e identificare eventuali somiglianze dei temi trattati. L'analisi dei topic rivela che entrambi gli intervistati trattano una gamma di temi che riflettono sia la loro vita personale che professionale. Si evince che entrambi parlano della famiglia e delle relazioni personali, degli aspetti finanziari e del loro successo come personaggi. Si differenziano nel fatto che Basalari si concentra sulle discoteche e sulla vita notturna, mentre Balti sui social media e sul mondo della moda e dello spettacolo.

```

def calculate_coherence(self):
    """Metodo che calcola il livello di coerenza complessivo dei topic generati dal modello LDA. La coerenza di un topic è una misura di quanto le parole che compongono il topic siano semanticamente simili tra loro. Un punteggio di coerenza più alto indica che le parole all'interno di un topic sono più semanticamente correlate."""
    tokens = self.df['Tokens'].tolist()
    dict = corpora.Dictionary(tokens)
    topics = [[token for token, weight in self.lda_model.show_topic(topic_id)] for topic_id in range(self.lda_model.num_topics)]
    coherence_model = CoherenceModel(topics=topics, texts=tokens, dictionary=dict, coherence='c_v')
    coherence_score = coherence_model.get_coherence()
    return coherence_score

```

Attraverso il metodo **calculate\_coherence** si calcola il livello di coerenza dei topic generati da un modello LDA (Latent Dirichlet Allocation). La coerenza di un topic misura quanto le parole che lo compongono siano semanticamente simili tra loro, e un punteggio di coerenza più alto indica che le parole all'interno del topic sono più correlate semanticamente. Il punteggio va da 0 a 1. Per l'intervista a Steven Basalari la coerenza dei topic è uguale a 0.388350035304229 e per Bianca Balti è uguale a 0.4253803064229412.

```

def compare_dominant_topics(self, other_model):
    """Metodo che serve per eseguire l'analisi comparativa dei topic dominanti tra due istanze di TopicModel. Utilizza il test del Chi-quadrato per valutare se c'è una relazione tra i testi delle interviste e i topic dominanti."""

    # le liste di token vengono convertite in bag-of-words
    bow_corpus_1 = [self.lda_model.id2word.doc2bow(doc) for doc in self.df['Tokens']]
    bow_corpus_2 = [other_model.lda_model.id2word.doc2bow(doc) for doc in other_model.df['Tokens']]

    # si ottengono i topic dominanti per ogni documento
    doc_lda_1 = [self.lda_model.get_document_topics(bow) for bow in bow_corpus_1]
    doc_lda_2 = [other_model.lda_model.get_document_topics(bow) for bow in bow_corpus_2]
    dominant_topics_1 = [max(topic, key=lambda x: x[1])[0] for topic in doc_lda_1]
    dominant_topics_2 = [max(topic, key=lambda x: x[1])[0] for topic in doc_lda_2]

    # viene creato un DataFrame con i testi e i topic dominanti
    data = {'text': ['doc1'] * len(dominant_topics_1) + ['doc2'] * len(dominant_topics_2),
            'dominant_topic': dominant_topics_1 + dominant_topics_2}
    df = pd.DataFrame(data)

    # viene applicato il test del Chi-quadrato
    contingency_table = pd.crosstab(df['text'], df['dominant_topic'])
    chi2, p_val, dof, expected = chi2_contingency(contingency_table)

    print("\nAnalisi comparativa dei topic dominanti:")
    print("Statistica del test del Chi-quadrato:", chi2)
    print("P-value:", p_val)

    # interpretazione dei risultati
    if p_val < 0.05:
        print("C'è una relazione significativa tra i testi e i topic dominanti.")
    else:
        print("Non c'è una relazione significativa tra i testi e i topic dominanti.")

```

Il metodo **compare\_dominant\_topics** serve per confrontare i topic dominanti tra le due istanze di **TopicModel**. Viene usato il parametro **other\_model** per usare due istanze all'interno di un metodo della classe. I testi inizialmente preprocessati vengono convertiti in bag-of-words. Attraverso il modello LDA vengono estratti i topic dominanti e per ciascun documento viene individuato il topic con la massima probabilità di appartenenza. Viene creato un DataFrame che contiene i testi dei documenti e i rispettivi topic dominanti per entrambi i modelli per facilitare l'analisi comparativa. Attraverso il test Chi-quadro si valuta se c'è una

relazione significativa tra i testi delle interviste e i topic dominanti. Innanzitutto viene creata una tabella di contingenza grazie al DataFrame definito che mostra la frequenza dei topic dominanti per ciascun testo, e su di essa si applica il test. I risultati indicano la statistica del test e il p-value che se è minore di 0.05 significa che c'è un relazione significativa tra i topic dominanti.

```
Analisi comparativa dei topic dominanti:
Statistica del test del Chi-quadrato: 36.47316053038837
P-value: 2.312499731530777e-07
C'è una relazione significativa tra i testi e i topic dominanti.
```

La statistica del test del Chi-quadrato è 36.47316053038837, questo valore misura la discrepanza tra le frequenze osservate dei topic dominanti nei testi delle due istanze di TopicModel e le frequenze attese se non ci fosse alcuna relazione tra i testi e i topic dominanti. Il p-value è 2.312499731530777e-07, valore bassissimo e molto inferiore ad alpha, quindi indica che c'è una relazione significativa tra i testi e i topic dominanti. Ciò significa che le differenze osservate nei topic dominanti tra i testi delle due istanze di TopicModel non sono dovute al caso, ma indicano effettivamente una differenza strutturale nei topic estratti dai due modelli.

```
def run_topic_analysis(self):
    """Metodo che richiama gli altri metodi, stampa il livello di coerenza dei topic per ogni intervista e crea un file CSV in cui a ogni frase viene accostato il topic con la probabilità più alta"""

    self.preprocess_texts()
    self.lda()
    self.save_visualization()
    self.topic_plot()
    self.word_cloud()

    # viene mandato a schermo il livello di coerenza di ciascun testo
    coherence_score = self.calculate_coherence()
    print(f"Coerenza dei Topic: {coherence_score}")
```

Nel metodo **run\_topic\_analysis** vengono richiamati i vari metodi della classe e viene mandato a schermo il livello di coerenza per ogni topic (definito nel metodo `calculate_coherence`).

```
# percorso dei file csv da cui verrà estratta, tramite pandas, la colonna "Frase" per l'analisi dei topic
file_path_1 = r'C:\Users\39392\Desktop\progettoWebAnalytics\file_csv\StevenBasalari_model_large.csv'
file_path_2 = r'C:\Users\39392\Desktop\progettoWebAnalytics\file_csv\BiancaBalti_model_large.csv'

# cartella di riferimento in cui verranno salvate le immagini png e html (creata se non esiste)
output_folder = r'C:\Users\39392\Desktop\progettoWebAnalytics\risultati_analisi_lda'
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Attraverso un messaggio si fa riferimento al file csv che si sta analizzando
# Creazione istanza per il file specificato e la cartella di riferimento per i risultati dell'analisi dei topic
# Chiamata del metodo run_topic_analysis() che permette attraverso tutta una serie di chiamate di altri metodi
# della classe lo sviluppo del codice
print("\nAnalisi Topic per l'intervista a Steven Basalari:")
topic_model_1 = TopicModel(file_path_1, output_folder)
topic_model_1.run_topic_analysis()

print("\nAnalisi Topic per l'intervista a Bianca Balti:")
topic_model_2 = TopicModel(file_path_2, output_folder)
topic_model_2.run_topic_analysis()

# Chiamata del metodo compare_dominant_topics dell'istanza topic_model_1 della classe TopicModel: passa come
# argomento l'istanza topic_model_2. Il metodo esegue un'analisi comparativa dei topic dominanti tra i due testi
# rappresentati dalle due istanze di TopicModel, utilizzando il test del Chi-quadrato.
topic_model_1.compare_dominant_topics(topic_model_2)
```

In questo blocco di codice si esegue l'analisi dei topic chiamando la classe **TopicModel**. Come parametri delle istanze vengono definiti i file CSV, tramite percorso specifico, contenenti le frasi dei testi delle interviste, e la cartella di output, che viene creata se non esiste già, in cui verranno salvati i risultati dell'analisi. Per ogni istanza verrà chiamato il metodo **run\_topic\_analysis**, che richiama gli altri metodi necessari per l'analisi. Infine, verrà chiamato il metodo **compare\_dominant\_topics** per confrontare i topic dominanti delle interviste.

## 7) Analisi comparativa sul sentiment, sulle emozioni e sulla coerenza dei topic dei testi delle interviste

Nello script **AnalisiComparativaInterviste.py** è presente il codice che permette di fare delle analisi statistiche sul sentiment, sulle emozioni e sulla coerenza dei topic delle frasi dei testi delle due interviste, presenti nei file CSV specifici per ogni intervista.

```
def t_test_sentiment_negativo(BiancaBalti_csv, StevenBasalari_csv):
    """Questa funzione esegue un t-test per confrontare i sentiment negativi tra due interviste. Passa come parametri, tramite percorso, i file CSV contenenti i dati delle due interviste analizzate"""

    df_BiancaBalti = pd.read_csv(BiancaBalti_csv)
    df_StevenBasalari = pd.read_csv(StevenBasalari_csv)

    # esecuzione del t-test per confrontare i sentiment negativi tra le due interviste
    t_stat, p_value = ttest_ind(df_BiancaBalti['Sentiment Negativo (%)'], df_StevenBasalari['Sentiment Negativo (%)]],
                                equal_var=False)

    # interpretazione del risultato
    alpha = 0.05 # Livello di significatività
    if p_value < alpha:
        risultato_sentiment = "Esiste una differenza significativa nei sentiment negativi tra le due interviste."
    else:
        risultato_sentiment = "Non c'è una differenza significativa nei sentiment negativi tra le due interviste."

    return t_stat, p_value, risultato_sentiment
```

La funzione **t\_test\_sentiment\_negativo** esegue un t-test per confrontare i sentiment negativi delle due interviste, presi dalla colonna specifica in cui risiedono i dati numerici sul sentiment negativo. Viene eseguito un test t a due campioni utilizzando **ttest\_ind** dalla libreria **scipy.stats**. Il parametro **equal\_var=False** specifica che si assume che le varianze delle due popolazioni non siano uguali.

```
Statistiche del t-test per il sentiment negativo:
Statistica t: -0.49658481938080684
Valore p_value: 0.6195568794538975
Non c'è una differenza significativa nei sentiment negativi tra le due interviste.
```

La **statistica t** misura la dimensione della differenza relativa alla variazione nei dati campione. Il valore è -0.4966 e indica la differenza tra le medie dei sentiment negativi delle due interviste non è grande rispetto alla variazione all'interno dei campioni. Il valore **p-value** è la probabilità di osservare un valore della statistica t almeno estremo quanto quello calcolato, assumendo che l'ipotesi nulla sia vera. Il suo valore è uguale a 0.6195568794538975. Il **livello di significatività ( $\alpha$ )** è 0.05 (5%). Se il valore del p-value è inferiore al livello di significatività si rigetta l'ipotesi nulla e indica che c'è una differenza significativa tra i sentiment negativi delle

due interviste. In questo caso è superiore al livello di significatività per cui non c'è una differenza significativa nei sentimenti negativi tra le due interviste.

```
def confronto_emozioni_predette(BiancaBalti_csv, StevenBasalari_csv):
    """Questa funzione confronta le distribuzioni delle emozioni predette tra due interviste utilizzando il test del chi-quadro. Passa come parametri, tramite percorso, i file CSV contenenti i dati delle due interviste analizzate"""

    df_BiancaBalti = pd.read_csv(BiancaBalti_csv)
    df_StevenBasalari = pd.read_csv(StevenBasalari_csv)

    # conteggio delle occorrenze delle emozioni predette per ciascuna intervista
    conteggio_emozioni_BiancaBalti = Counter(df_BiancaBalti['Emozione Predetta'].apply(eval).sum())
    conteggio_emozioni_StevenBasalari = Counter(df_StevenBasalari['Emozione Predetta'].apply(eval).sum())

    # creazione dei dataframe per le distribuzioni di frequenza delle emozioni predette
    df_conteggio_BiancaBalti = pd.DataFrame.from_dict(conteggio_emozioni_BiancaBalti, orient='index',
                                                       columns=['BiancaBalti'])
    df_conteggio_StevenBasalari = pd.DataFrame.from_dict(conteggio_emozioni_StevenBasalari, orient='index',
                                                       columns=['StevenBasalari'])

    # unione dei dataframe delle distribuzioni delle emozioni predette
    df_comparativo_emozioni = df_conteggio_BiancaBalti.join(df_conteggio_StevenBasalari, how='outer').fillna(0)
    # esecuzione del test del chi-quadro per confrontare le distribuzioni delle emozioni predette
    chi2, p_value, _, _ = chi2_contingency(df_comparativo_emozioni)

    # Interpretazione del risultato
    alpha = 0.05 # Livello di significatività
    if p_value < alpha:
        risultato_emozioni = "Esiste una differenza significativa nelle distribuzioni delle emozioni predette tra le due interviste."
    else:
        risultato_emozioni = "Non c'è una differenza significativa nelle distribuzioni delle emozioni predette tra le due interviste."
    return chi2, p_value, risultato_emozioni, df_comparativo_emozioni
```

La funzione **confronto\_emozioni\_predette** confronta le distribuzioni delle emozioni predette delle interviste utilizzando il test del chi-quadro. Dai file CSV di ogni intervista attraverso il metodo **Counter** vengono contate le occorrenze di ogni emozione predetta. Con l'utilizzo della funzione **apply(eval). sum()** le emozioni sono codificate come liste per poi essere sommate. Successivamente i conteggi sono convertiti in DataFrame pandas e vengono uniti in una tabella comparativa delle emozioni predette, riempiendo eventuali valori mancanti con 0. Viene eseguito il test del chi-quadro per confrontare le distribuzioni delle emozioni predette con la **funzione chi2\_contingency**. Il **livello di significatività ( $\alpha$ )** è impostato a 0.05 (5%) e se il p-value è minore di alpha vuol dire che esiste una differenza significativa nelle distribuzioni delle emozioni predette tra le due interviste.

```
Risultati del test del chi-quadro per confronto delle emozioni predette:  
Chi-quadro: 0.23774907098843467  
Valore p_value: 0.9712768374590375  
Non c'è una differenza significativa nelle distribuzioni delle emozioni predette tra le due interviste.
```

La statistica del chi-quadro valuta la differenza tra le distribuzioni osservate e quelle previste dall'ipotesi nulla. Un valore maggiore indica una maggiore discrepanza tra queste distribuzioni. I risultati della funzione di codice indicano che il **valore chi-quadro** è di 0.2377, valore piuttosto basso, che suggerisce che le distribuzioni osservate delle emozioni predette nelle due interviste sono molto simili. Il **p-value** invece indica la probabilità di ottenere una statistica chi-quadro quanto quella osservata se si assume il fatto che l'ipotesi nulla sia vera. Essendo un valore molto elevato (0.9712768374590375) rispetto ad alpha, questo suggerisce che non c'è una differenza significativa tra le distribuzioni delle emozioni predette nelle due interviste e quindi l'ipotesi nulla viene rigettata.

```

def confronto_coerenza_topic(StevenBasalari_coerenza, BiancaBalti_coerenza):
    """Questa funzione confronta la coerenza dei topic tra le due interviste utilizzando per ciascuna di esse un numero che rappresenta il livello di coerenza complessivo dei topic. Il livello di coerenza va da 0 a 1. La funzione indica l'intervista che ha un maggiore livello di coerenza"""

    if StevenBasalari_coerenza > BiancaBalti_coerenza:
        risultato_coerenza = "I topic di Steven Basalari sono più coerenti rispetto a quelli di Bianca Balti"
    elif StevenBasalari_coerenza < BiancaBalti_coerenza:
        risultato_coerenza = "I topic di Bianca Balti sono più coerenti rispetto a quelli di Steven Basalari"
    else:
        risultato_coerenza = "I topic di Bianca Balti sono coerenti quanto quelli di Steven Basalari"

    return f"\n{risultato_coerenza}"

# il livello di coerenza per ciascuna intervista preso dal file TopicModelling.py, passato come parametro degli # argomenti della funzione 'confronto_coerenza_topic' che viene chiamata
print(confronto_coerenza_topic( StevenBasalari_coerenza: 0.3883500353042295, BiancaBalti_coerenza: 0.4253803064229412))

```

Nella funzione **confronto\_coerenza\_topic** vengono riportati i livelli di coerenza dei topic per ognuna delle due interviste effettuati nello script **TopicModelling.py**, indicando qual è quello maggiore o se sono uguali. Risultano più coerenti i topic dell'intervista di Bianca Balta anche se i numeri sono molto vicini tra di loro.

## 8) Conclusioni sulla prima parte del progetto

Nella prima parte del progetto ho incontrato varie difficoltà e potenziali fonti di inattendibilità nei risultati, dovute al fatto che i testi analizzati derivano dalla trascrizione automatica di audio in lingua italiana, contaminata da espressioni in dialetto e lingue straniere. I modelli speech-to-text come quello che ho utilizzato (Whisper), nonostante siano molto accurati ottengono molti errori che fanno rumore nel testo tra imprecisioni, parole mancanti o mal riconosciute che rendono meno efficiente i risultati delle analisi di sentiment, emozioni e topic.

Per migliorare le analisi sono stati applicati dei preprocessamenti ai testi come rimozione di punteggiatura, conversione in minuscolo, tokenizzazione e lemmatizzazione. Sono anche state aggiunte stopwords personalizzate durante l'analisi dei topic. Nonostante questi accorgimenti i modelli di analisi utilizzati, essendo addestrati molto probabilmente su testi puliti, potrebbero non funzionare altrettanto bene su testi "rumorosi" come le trascrizioni di audio, specialmente in lingua italiana contaminata. I risultati vanno presi con una certa precauzione, in quanto la qualità dei testi non è perfetta.

Seguendo gli obiettivi che riguardano la prima parte del progetto è stata effettuata l'analisi del sentiment (positivo/negativo) e dell'emotion analysis (gioia, paura, rabbia, tristezza) su ogni singola frase dei testi audio delle due interviste selezionate dal podcast "One More Time di Luca Casadei". Sono stati creati dei grafici appositi che mostrano sia l'occorrenza del sentiment che delle emozioni sulle frasi dei testi, e come varia il sentiment e le emozioni durante l'intervista attraverso grafici in cui sono state costruite serie temporali. Si è applicato il topic modeling sui testi per estrarre i 5 topic principali per ogni intervista, visualizzandoli attraverso wordcloud, grafici a barre e interfacce interattive.

I risultati mostrano che in entrambe le interviste prevale un sentiment negativo e l'emozione predominante è la tristezza. I topic emersi spaziano su vari argomenti che vanno dalla famiglia al successo professionale. Le analisi comparative, attraverso test statisticci, hanno evidenziato che non ci sono differenze significative nei sentiment negativi e nella distribuzione delle emozioni predette tra le due interviste, e che non c'è una relazione significativa tra i testi e i topic dominanti.

## Seconda parte del progetto: analisi dei commenti estratti da YouTube

La seconda parte del progetto si occupa di effettuare un'analisi sui commenti estratti da video su YouTube. Nel branch chiamato "**analisi\_commenti\_videoYouTube**" vengono raccolti i dati e gli script necessari per effettuare: sentiment analysis, emotion analysis, topic modeling e un'analisi comparativa tra i commenti estratti dai video di alcune interviste selezionate di "One More Time". Verranno spiegati passo per passo i passaggi affrontati per completare la seconda parte del progetto. Durante lo sviluppo degli script ho utilizzato, dove mi è stato possibile, le stesse modalità di costruzione progettuale del branch "analisi\_AudioTesto\_videoYouTube".

### 1) Selezione di dieci video tra i più popolari del canale One More Time su YouTube

Per la seconda parte del progetto, ho selezionato dieci tra le interviste più commentate del canale di *One More Time* di *Luca Casadei* su YouTube. Ho scelto di concentrarmi sui commenti dei video di: Alex Cotoia, Andrea Presti, Benjamin Mascolo, Bianca Balti, Marco Baldini, Mario Maccione, Michele Morrone, Nicoletta Amato, Sara Tommasi e Susi Gallesi. I commenti totali per ogni intervista sono variabili e a parte l'intervista di Presti che raggiunge poco più di 450 commenti, le altre vanno oltre i 600, arrivando anche a più di 1100.

### 2) Creazione di uno script per l'estrazione dei commenti di video di YouTube

Nello script **YoutubeCommentScraper.py**, il codice è stato progettato per l'estrazione dei commenti delle interviste selezionate, i quali vengono salvati in file CSV specifici per ciascuna intervista. Per la costruzione del mio codice, adattandolo alle mie esigenze, ho seguito i consigli del video <https://www.youtube.com/watch?v=0FtcHjl5lmw>. Ho dovuto accedere a Google Cloud, ed entrare su "Api e Servizi Abilitati" e selezionare "YouTube Data API v3"<sup>11</sup> per procedere con l'abilitazione. Ciò mi ha permesso entrando su "Credenziali" di creare una chiave Api che permetterà al codice Python di estrarre i commenti per ogni video.

```
class ScraperComment:
    """Classe creata per estrarre i commenti di video su YouTube. Permette di interagire con l'API di YouTube e recuperare i commenti dei video. Definisce tre costanti: api_service_name, api_version e DEVELOPER_KEY. La prima è impostata su 'youtube' e indica il servizio con cui interagire, la seconda indica la versione API di YouTube e l'ultima indica la chiave di API di sviluppatore valida, necessaria per autenticare le richieste all'API di YouTube"""

    api_service_name = "youtube"
    api_version = "v3"
    DEVELOPER_KEY = "AIzaSyC06QCasjpaK7FOe3lPpFU-zEMr5tIn2Ew"
    youtube = googleapiclient.discovery.build(
        api_service_name, api_version, developerKey=DEVELOPER_KEY)

    def __init__(self, video_id, ospite):
        """Metodo costruttore che richiama per ogni oggetto della classe l'attributi ID del video di Youtube e nome dell'ospite dell'intervista"""

        self.video_id = video_id
        self.ospite = ospite
```

<sup>11</sup> Link del servizio YouTube Data API v3:

<https://console.cloud.google.com/apis/api/youtube.googleapis.com/metrics?project>

In questo pezzo di codice viene definita la classe **ScraperComment**, progettata per estrarre i commenti di video utilizzando l'API di YouTube. La classe ha degli attributi costanti che impostano **youtube** per indicare il servizio con cui interagire, **l'api\_version** che indica la versione dell'API di YouTube utilizzata e **DEVELOPER\_KEY** che è la chiave di API da sviluppatore valida, è necessaria per autenticare le richieste all'API di YouTube.

Nel metodo del costruttore vengono accettati due parametri: **video\_id** e **ospite** che rappresentano l'ID del video di YouTube e il nome dell'ospite dell'intervista. I valori vengono assegnati dalle istanze definite successivamente.

```
def get_comments_with_metadata(self):
    """Questo metodo recupera i commenti di un video YouTube insieme a metadati importanti, come il nome utente del commento, il testo di esso e la sua data di pubblicazione. Sfrutta la YouTube Data API per ottenere i dati richiesti in lotti (pagine) fino a esaurimento dei commenti disponibili"""

    comments_data = []
    request = self.youtube.commentThreads().list(
        part="snippet",
        videoId=self.video_id,
        maxResults=100
    )

    while request:
        response = request.execute()
        for item in response['items']:
            username = item['snippet']['topLevelComment']['snippet']['authorDisplayName']
            comment_text = item['snippet']['topLevelComment']['snippet']['textDisplay']
            comment_date = item['snippet']['topLevelComment']['snippet']['publishedAt']
            comments_data.append({'Username': username, 'Commento': comment_text, 'Data': comment_date})
        request = self.youtube.commentThreads().list_next(request, response)

    return comments_data
```

Il metodo della classe **get\_comments\_with\_metadata** è progettato per recuperare i commenti di un video di YouTube insieme a metadati importanti come il nome utente del commento, il testo e la data di pubblicazione del commento. Viene inizializzata una lista vuota che conterrà in seguito i commenti estratti con i metadati associati. Con l'oggetto **requests** viene utilizzato il metodo **commentThreads().list()** dell'oggetto youtube in cui vengono definiti dei parametri per specificare quale parte dei dati deve essere estrapolata, il codice identificativo del video dell'intervista e il numero massimo dei commenti da recuperare per ogni richiesta. Attraverso il ciclo while vengono recuperati tutti i commenti delle pagine di essi e ad ogni iterazione, viene eseguita la richiesta e la risposta viene salvata nella variabile **response**. Viene iterato attraverso gli elementi della risposta per estrarre le informazioni sui commenti e queste informazioni vengono poi aggiunte come un dizionario alla lista **comments\_data**. Attraverso il metodo il **metodo commentThreads().list\_next()** se ci sono pagine disponibili, viene ottenuta la richiesta successiva. Una volta terminate le pagine di commenti disponibili, viene restituita la lista **comments\_data** contenente i commenti estratti insieme ai metadati come output del metodo.

```

def save_comments_to_csv(self):
    """Metodo che salva i commenti dei video su YouTube e tutte le informazioni annesse mediante un DataFrame Pandas,
    che successivamente viene trasformato in un file CSV e salvato nella cartella file_commenti_csv."""

    comments_data = self.get_comments_with_metadata()
    comments_df = pd.DataFrame(comments_data)

    folder_name = "file_commenti_csv"
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)

    filename = f"{folder_name}/{self.ospite}.csv"
    comments_df.to_csv(filename, index=False)

    return (f"IL DataFrame dell'ospite {self.ospite} è stato salvato come {filename} "
           f"nella cartella file_commenti_csv\n")

```

Nel metodo **save\_comments\_to\_csv** i commenti e le relative informazioni estratti vengono salvati in un file CSV specifico per ogni intervista. Attraverso la lista ottenuta dal metodo precedente, viene creato un DataFrame pandas che permette di gestire i dati in modo strutturato per poi essere trasformato in un file CSV che si troverà nella cartella **file\_commenti\_csv**, che verrà creata se non esiste. Per ogni file salvato verrà mandato a schermo la creazione un messaggio che lo conferma.

```

# Creazione di istanze della classe ScraperComment in cui, per ogni video di cui si vogliono scaricare i metadati dei
# commenti, si passano come attributi gli ID dei video YouTube per identificarli e i nomi degli ospiti. Alla fine, si
# chiama il metodo che permette di salvare il DataFrame creato nella classe come file CSV.
scraper1 = ScraperComment(video_id="NU0e8EgEj-0", ospite="AlexCotoia").save_comments_to_csv()
scraper2 = ScraperComment(video_id="-szezlsJvYQ", ospite="BiancaBalti").save_comments_to_csv()
scraper3 = ScraperComment(video_id="nYYWoKzJNvE", ospite="BenjaminMascolo").save_comments_to_csv()
scraper4 = ScraperComment(video_id="NXcaaQ4u00k", ospite="MicheleMorrone").save_comments_to_csv()
scraper5 = ScraperComment(video_id="ZXSLNzQELKs", ospite="SusiGallesi").save_comments_to_csv()
scraper6 = ScraperComment(video_id="RWcXxpr00cm", ospite="MarcoBaldini").save_comments_to_csv()
scraper7 = ScraperComment(video_id="Uq96mBBzkbU", ospite="SaraTomasini").save_comments_to_csv()
scraper8 = ScraperComment(video_id="S3Yy0Zq6foA", ospite="NicolettaAmato").save_comments_to_csv()
scraper9 = ScraperComment(video_id="P2cqM9-t7u8", ospite="MarioMaccione").save_comments_to_csv()
scraper10 = ScraperComment(video_id="QZiC-Z5nE0W", ospite="AndreaPresti").save_comments_to_csv()

if __name__ == "__main__":
    # chiamata degli oggetti della classe
    print(scraper1, scraper2, scraper3, scraper4, scraper5,
          scraper6, scraper7, scraper8, scraper9, scraper10)

```

In questo pezzo di codice si istanzieranno della classe che passano i parametri che identificano il video YouTube con l'ID e con il nome dell'ospite dell'intervista. Ad ogni oggetto viene passato il metodo che salva i commenti con i suoi dati come file CSV. Successivamente vengono chiamate le istanze.

```

def remove_user_from_files(folder_path):
    """Dopo aver creato la classe ScraperComment, ho creato una funzione che prende la cartella contenente i file csv e ho tolto tutti i commenti pubblicati dalla pagina ufficiale in quanto non servono ai fini del progetto, infine ho aggiornato i file"""

    user_to_remove = "@OneMoreTimePodcast" # utente pagina ufficiale
    for file_name in os.listdir(folder_path):

        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path):
            with open(file_path, "r", encoding="utf-8") as file:
                reader = csv.reader(file)
                content = list(reader)

                header = content[0]
                new_content = [header]
                for row in content[1:]:
                    if user_to_remove not in row[0]:
                        new_content.append(row)

                new_file_path = os.path.join(folder_path, file_name)
                with open(new_file_path, "w", encoding="utf-8", newline='') as new_file:
                    writer = csv.writer(new_file)
                    writer.writerows(new_content)

            print(f"Il contenuto del file {file_name} è stato modificato e salvato come {new_file_path}")

remove_user_from_files(folder_path=r"C:\Users\39392\Desktop\progettoWebAnalytics\file_commenti_csv")

```

Dopo la costruzione della classe con la cartella contenente i file CSV, ho creato una funzione che ha eliminato tutti i commenti della pagina ufficiale e ho aggiornato tutti i file CSV.

### 3) Analisi del sentiment e delle emozioni sui commenti delle interviste

Lo script **SentimentEmotionAnalysis\_Comments.py** è stato progettato per pulire i commenti presi dai file CSV ed eseguire su di loro un'analisi del sentiment e delle emozioni.

```

def clean_comment(comment):
    """Funzione che controlla i commenti dei file csv, rimuove sequenze e tag HTML (restituisce il carattere originale in alcuni casi), e rimuove e le emoji presenti"""

    EMOJI_PATTERN = re.compile("["
        "\U0001F1E0-\U0001F1FF" # flags (iOS)
        "\U0001F300-\U0001F5FF" # symbols & pictographs
        "\U0001F600-\U0001F64F" # emoticons
        "\U0001F680-\U0001F6FF" # transport & map symbols
        "\U0001F700-\U0001F77F" # alchemical symbols
        "\U0001F780-\U0001F7FF" # Geometric Shapes Extended
        "\U0001F800-\U0001F8FF" # Supplemental Arrows-C
        "\U0001F900-\U0001F9FF" # Supplemental Symbols and Pictographs
        "\U0001FA00-\U0001FA6F" # Chess Symbols
        "\U0001FA70-\U0001FAFF" # Symbols and Pictographs Extended-A
        "\U00002702-\U00002780" # Dingbats
        "\U000024C2-\U0001F251"
    "]")

    if isinstance(comment, str):
        comment = comment.replace(_old: """, _new: "").replace(_old: "&#39;", _new: "'")
        comment = re.sub(pattern: r'<[^>]*>', repl: '', comment)
        comment = EMOJI_PATTERN.sub(repl: r'', comment) # per evitare che le parole attaccate alle emoji si attacchino
        comment = comment.replace(_old: " ", _new: "").strip()

        if len(comment) < 2:
            comment = None

    return comment

```

Attraverso la funzione **clean\_comment** i commenti, definiti nella funzione successiva, vengono puliti rimuovendo tag HTML, sequenze di caratteri HTML ed emoji. Viene usata una regex per rimuovere le emoji, vengono sostituite sequenze HTML con i rispettivi caratteri oppure totalmente eliminate come i tag HTML. Vengono rimossi gli spazi duplicati e quelli iniziali/finali. E se il commento è minore di due lettere viene impostato a None. Alla fine, vengono restituiti i commenti.

```
def process_files(path):
    """Funzione che elabora i file CSV del path definito, dove si trovano i file con i metadati scaricati dei video delle interviste. Vengono eliminati le righe che hanno i commenti vuoti e le modifiche vengono salvate nei file CSV di origine"""

    files = os.listdir(path)
    for file in files:
        if file.endswith('.csv'):
            file_path = os.path.join(path, file)

            df = pd.read_csv(file_path)
            df['Commento'] = df['Commento'].apply(clean_comment)
            df = df.dropna(subset=['Commento']) # elimina i commenti vuoti

            df.to_csv(file_path, index=False)

    print(f"I commenti del file {file} sono stati modificati")
```

Attraverso questa funzione vengono elaborati i file CSV richiamati con il metodo listdir in cui viene passato il percorso della cartella che li contiene. Attraverso un iterazione vengono richiamati i commenti presenti su ciascun file e nel caso i commenti siano vuoti vengono eliminati.

```
def sentiment_and_emotion_analysis(path):
    """La funzione permette l'analisi del sentimento e delle emozioni dei commenti di video su YouTube. Sono stati usati modelli pre-addestrati che fanno predizione del sentimento (positiva e negativa) e delle emozioni (indicate come: joy, sadness, fear, anger). I dati per ciascun commento verranno salvati nel file csv di origine"""

    # Caricamento del modello per l'analisi del sentimento del testo in lingua italiana
    sentiment_classifier = pipeline(task="text-classification", model='MilaNLProc/feel-it-italian-sentiment', top_k=2)

    # Caricamento della funzione per l'analisi delle emozioni del testo in lingua italiana
    emotion_classifier = EmotionClassifier()

    for file_name in os.listdir(path):
        file_path = os.path.join(path, file_name)
        if os.path.isfile(file_path):
            with open(file_path, "r", encoding="utf-8") as file:
                reader = csv.reader(file)
                content = list(reader)

                header = content[0]
                header.extend(["Sentiment Positivo", "Sentiment Negativo", "Emozione Predetta"])
                new_content = [header]

                header = content[0]
                header.extend(["Sentiment Positivo", "Sentiment Negativo", "Emozione Predetta"])
                new_content = [header]

                for row in content[1:]:
                    segment = row[1]
                    try:
                        sentiment_predictions = sentiment_classifier(segment)[0]
                        pos_score = next((item['score'] for item in sentiment_predictions if item['label'] == 'positive'), 0)
                        neg_score = next((item['score'] for item in sentiment_predictions if item['label'] == 'negative'), 0)
                        predicted_emotion = emotion_classifier.predict([segment])
                        row.extend([pos_score, neg_score, predicted_emotion])
                        new_content.append(row)
                    except Exception as e:
                        print(f"Errore durante l'analisi della frase: {e}")

```

```

    # aggiornamento del file
    new_file_path = os.path.join(path, file_name)
    with open(new_file_path, "w", encoding="utf-8", newline='') as new_file:
        writer = csv.writer(new_file)
        writer.writerows(new_content)

    print(f"Il contenuto del file {file_name} è stato modificato e salvato come {new_file_path}")

```

In questi pezzi di codice viene rappresentata la funzione che permette l'analisi del sentiment e delle emozioni, che ha come parametro il percorso della cartella che contiene i file CSV.

Per queste analisi vengono usati gli stessi modelli utilizzati nel branch precedente. Effettuata l'analisi coi modelli preaddestrati, il file CSV originale avrà in aggiunta delle colonne che rappresenteranno per ogni commento un sentiment positivo, negativo e l'emozione predetta. Alla fine della funzione viene mandato a schermo il messaggio che implica che il file CSV specifico è stato modificato con un nuovo file con le aggiunte.

```

# Percorso della cartella che contiene i file CSV con tutti i commenti dei video delle interviste su Youtube
path = r'C:\Users\39392\Desktop\progettoWebAnalytics\file_commenti_csv'

# chiamata della funzione che si occupa dell'elaborazione dei file, che ha parametro il path della cartella con
# i file CSV che hanno i dati estratti
process_files(path)

# chiamata della funzione che permette l'analisi del sentiment e delle emozioni dei commenti
sentiment_and_emotion_analysis(path)

```

In questo blocco di codice viene definito il percorso della cartella con i vari file CSV e la chiamata delle funzioni che la processazione dei file e per le analisi del sentiment e delle emozioni, che passano come parametro il percorso appena definito.

#### 4) Creazione grafici sul sentiment e sulle emozioni, e analisi comparativa dei commenti delle interviste

Nel file script **PlotAndTestForCommentsOnInterviews.py** è contenuto del codice che ha come finalità la creazione di grafici per rappresentare il sentiment positivo/negativo e l'occorrenza delle emozioni predette nei commenti estratti dai video su YouTube delle interviste selezionate, e lo svolgimento di test statistici per determinare se ci sono differenze significative nei sentiment negativi tra le interviste e se esiste un'associazione significativa tra il tipo di emozione e il tipo di sentiment. I grafici delle immagini vengono riportati in una cartella **grafici\_immagini\_commentiYT**.

```

def plot_sentiment(directory_path):
    """Funzione che crea un grafico a barre accostate che indica per ciascuna intervista il sentiment medio negativo e
    positivo (in percentuale) dei commenti per ciascuna intervista"""

    file_names, positive_sentiments, negative_sentiments = [], [], []
    for filename in os.listdir(directory_path):
        if filename.endswith(".csv"):
            filepath = os.path.join(directory_path, filename)
            df = pd.read_csv(filepath)
            file_names.append(os.path.splitext(filename)[0])
            positive_sentiments.append(df['Sentiment Positivo'].mean() * 100)
            negative_sentiments.append(df['Sentiment Negativo'].mean() * 100)

    results_df = pd.DataFrame({'File': file_names, 'Sentiment Positivo': positive_sentiments, 'Sentiment Negativo': negative_sentiments})

```

```

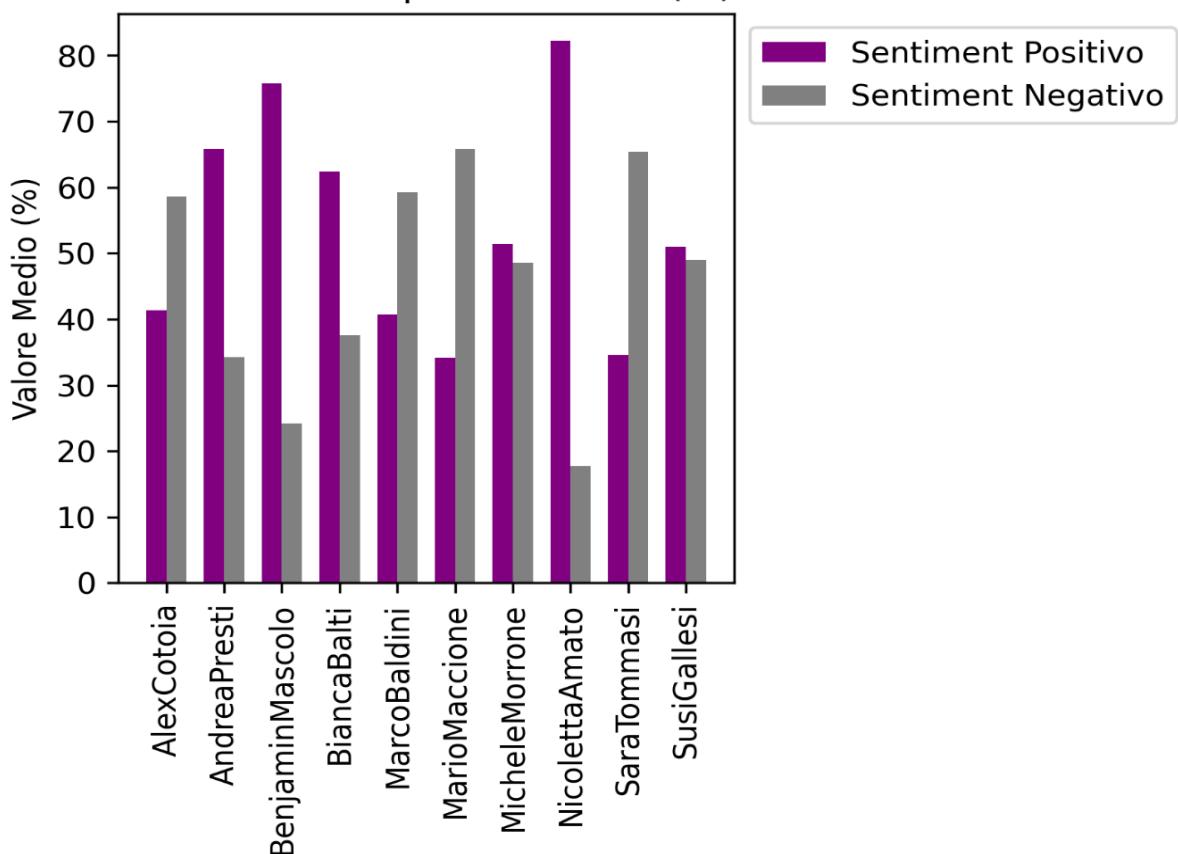
x = range(len(file_names))
width = 0.35
fig, ax = plt.subplots()
bars1 = ax.bar(x, results_df['Sentiment Positivo'], width, label='Sentiment Positivo', color='purple')
bars2 = ax.bar([i + width for i in x], results_df['Sentiment Negativo'], width, label='Sentiment Negativo', color='grey')
ax.set_ylabel('Valore Medio (%)')
ax.set_title('Media del sentiment positivo e negativo\n dei commenti per intervista (%)')
ax.set_xticks([i + width / 2 for i in x])
ax.set_xticklabels(file_names, rotation=90)
ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
fig.tight_layout(rect=[0, 0, 0.85, 1])

plt.show()
fig.savefig(os.path.join('grafici_immagini_commentiYT', 'sentiment_per_intervista.png'), dpi=300, bbox_inches='tight')

```

In questi blocchi di codice viene definita una funzione **plot\_sentiment** che ha il fine di creare un grafico a barre, mostrando i sentiment medi positivi e negativi (in percentuale) dei commenti per ogni intervista presente nei file CSV, definiti dagli altri script. Ognuna di queste è identificata con il nome dell'ospite di Luca Casadei. Per la costruzione dei grafici, vengono inizializzate tre liste vuote che conterranno: i nomi degli intervistati (presi dai titoli dei file CSV originali da cui si prenderanno i dati), i sentiment positivi e negativi. Di questi ultimi si farà la media, valutata in percentuale per il grafico. Il sentiment positivo sarà di colore viola, mentre quello negativo di colore grigio.

**Media del sentiment positivo e negativo  
dei commenti per intervista (%)**



7. *Grafico a barre accostate che rappresenta il sentiment medio positivo e negativo (calcolato in %) dei commenti di ogni intervista di "One More Time" selezionata*

Dal grafico si può notare come, nella maggior parte dei casi, i sentiment positivi prevalgano su quelli negativi. Nicoletta Amato e Benjamin Mascolo sono gli ospiti che presentano un sentiment positivo molto più alto rispetto a quello negativo (entrambi hanno circa l'80% di sentiment positivi). Mario Maccione e Sara Tommasi, invece, sono gli ospiti per i quali predomina maggiormente il sentiment negativo rispetto a quello positivo. Nelle interviste di Michele Morrone e Susi Gallesi, invece, i sentiment positivi e negativi sono più o meno equivalenti.

```
def plot_emotions(directory_path):
    """Funzione che crea un grafico a barre accostate per indicare l'occorrenza (in percentuale) di una certa emozione all'interno dei commenti per ciascuna intervista"""

    emotions = ['joy', 'sadness', 'anger', 'fear']
    file_names = []
    emotion_percentages = {emotion: [] for emotion in emotions}

    for filename in os.listdir(directory_path):
        if filename.endswith(".csv"):
            filepath = os.path.join(directory_path, filename)
            df = pd.read_csv(filepath)
            file_names.append(os.path.splitext(filename)[0])

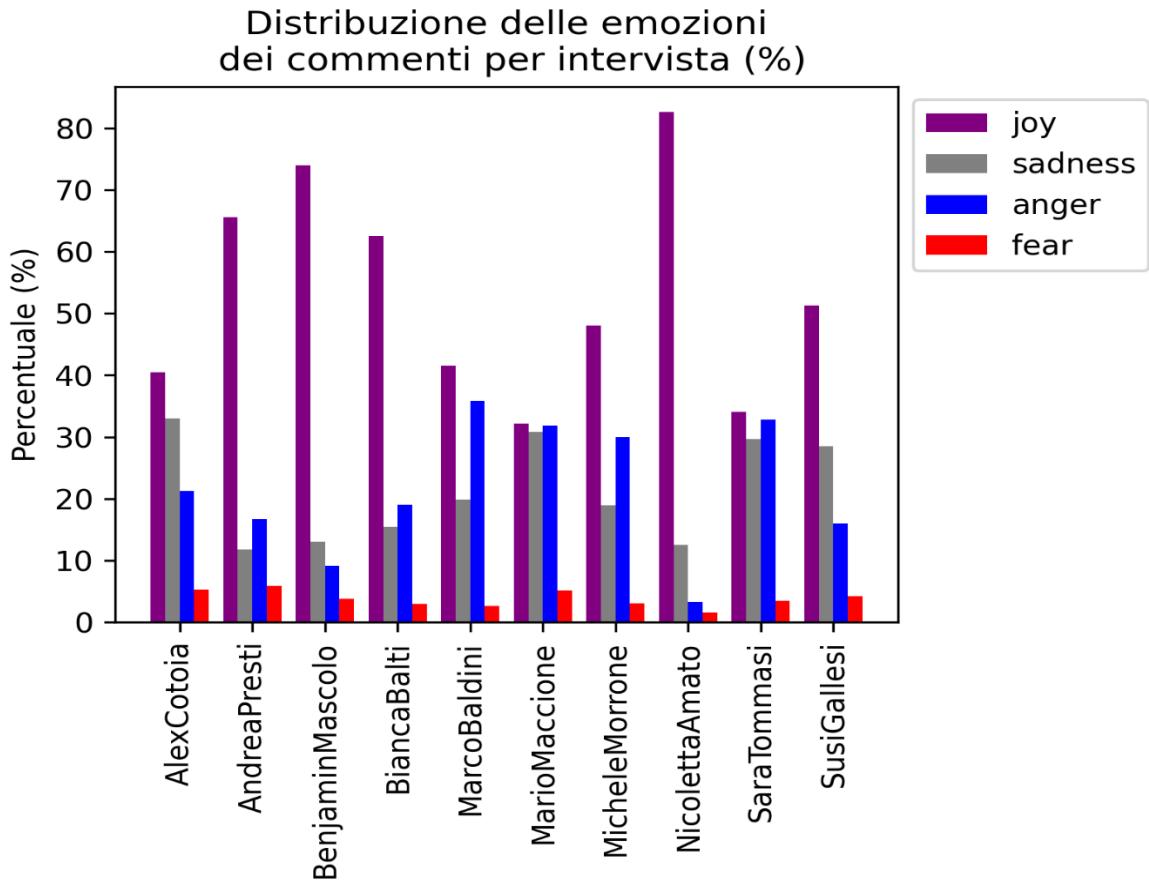
            emotion_count = df['Emozione Predetta'].str.strip("[]").str.replace("'", "").str.split(
                ",").explode().value_counts()
            total_comments = len(df)
            for emotion in emotions:
                percentage = (emotion_count.get(emotion, 0) / total_comments) * 100
                emotion_percentages[emotion].append(percentage)

    results_df = pd.DataFrame(emotion_percentages, index=file_names)

    fig, ax = plt.subplots()
    x = range(len(file_names))
    width = 0.2
    colors = {'joy': 'purple', 'sadness': 'grey', 'anger': 'blue', 'fear': 'red'}
    for i, emotion in enumerate(emotions):
        ax.bar([p + width * i for p in x], results_df[emotion], width, label=emotion, color=colors[emotion])
    ax.set_ylabel('Percentuale (%)')
    ax.set_title('Distribuzione delle emozioni\n dei commenti per intervista (%)')
    ax.set_xticks([p + width * 1.5 for p in x])
    ax.set_xticklabels(file_names, rotation=90)
    ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
    fig.tight_layout(rect=[0, 0, 0.85, 1])

    plt.show()
    fig.savefig(os.path.join('grafici_immagini_commentiYT', 'emozioni_per_intervista.png'), dpi=300, bbox_inches='tight')
```

In questo blocco di codice viene definita la funzione **plot\_emotions** che crea un grafico a barre accostate per visualizzare la percentuale di occorrenza di determinate emozioni (gioia, tristezza, rabbia, paura) sui commenti di ciascuna intervista, contenuti nei file CSV specifici. Viene inizializzata una lista per fornire i nomi degli ospiti delle interviste e creato un dizionario per memorizzare le percentuali delle emozioni per ciascuna intervista. Vengono definiti anche i colori per ogni emozione predetta, che sarà rappresentata da una barra specifica per ogni emozione su ogni intervista. Anche questo grafico verrà salvato in una cartella apposita.



8. *Grafico a barre accostate che rappresentano la distribuzione delle emozioni (calcolate in %) sui commenti per ogni intervista selezionata di "One More Time"*

Visualizzando il grafico della distribuzione delle emozioni rilevate dai commenti su ogni intervista, si può notare che l'emozione predominante nei commenti per la maggior parte delle interviste è la gioia, rappresentata dalle barre più alte di colore viola. Le eccezioni si trovano nelle interviste di Mario Maccione e Sara Tommasi, dove le percentuali di gioia, tristezza e rabbia sono più o meno simili. La paura, rappresentata dalle barre rosse, è l'emozione che in tutte le interviste ha un punteggio percentuale più basso rispetto alle altre emozioni.

```
def test_negative_sentiment(directory_path):
    """Funzione che analizza i sentiment negativi delle interviste, li combina in un unico DataFrame, esegue alcune operazioni di pre-elaborazione sui dati ed effettua un'analisi della varianza (ANOVA) per determinare se ci sono differenze significative nei sentiment negativi tra i file considerati"""

    dfs = []
    for filename in os.listdir(directory_path):
        if filename.endswith(".csv"):
            filepath = os.path.join(directory_path, filename)
            df = pd.read_csv(filepath)
            df['File'] = os.path.splitext(filename)[0]
            df['Sentiment_Negativo'] = df['Sentiment Negativo'].astype(str).str.replace('[^0-9.-]', '',
                                                                                           regex=True).astype(float)
            dfs.append(df)
```

```

combined_df = pd.concat(dfs, ignore_index=True)
model = ols('Sentiment_Negativo ~ C(File)', data=combined_df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
print("\nInterpretazione del risultato dell'ANOVA sui sentiment negativi delle interviste:")
if anova_table['PR(>F)'][0] < 0.05:
    print("Esiste una differenza significativa nei sentiment negativi tra i commenti delle interviste prese in "
          "considerazione.")
else:
    print("Non c'è evidenza sufficiente per concludere che ci siano differenze significative nei sentiment negativi "
          "tra le interviste considerate.")

```

La funzione **test\_negative\_sentiment** è progettata per effettuare un'analisi sui sentiment negativi dei commenti presenti nei file CSV delle interviste. Le informazioni sui sentiment negativi vengono estratte e, successivamente, viene eseguita un'analisi della varianza (ANOVA) per valutare se esistono differenze significative nei sentiment negativi tra le diverse interviste. Attraverso questa analisi statistica si determina se le variazioni osservate nei sentiment negativi possono essere attribuite alla casualità o se sono statisticamente significative. Completata l'analisi vengono interpretati i risultati dell'ANOVA. Se la probabilità (valore p-value) associata al test ANOVA è inferiore a un certo livello di significatività (impostato a 0.05), si conclude che esistono differenze significative nei sentiment negativi tra le interviste esaminate. Altrimenti, si conclude che non ci sono prove sufficienti per sostenere l'esistenza di tali differenze.

	sum_sq	df	F	PR(>F)
C(File)	179.294881	9.0	92.512953	1.802564e-102
Residual	1336.609628	6207.0	NaN	NaN

Interpretazione del risultato dell'ANOVA sui sentiment negativi delle interviste:  
Esiste una differenza significativa nei sentiment negativi tra i commenti delle interviste prese in considerazione.

I risultati dell'analisi indicano che c'è una differenza significativa nei sentiment negativi tra i commenti delle diverse interviste considerate. Questo risultato è supportato da un valore basso del p-value (indicato come PR(>F)), che è praticamente zero. Ciò suggerisce che se l'ipotesi nulla fosse vera, ossia che la differenza nei sentiment negativi tra le interviste fosse dovuta al caso, la probabilità di osservare una differenza così significativa sarebbe estremamente bassa. In conclusione, il test ANOVA ci dice che le differenze nei sentiment negativi osservati non sono casuali, ma probabilmente riflettono vere differenze tra le interviste.

```

def test_emotions(directory_path):
    """Funzione che analizza l'associazione tra emozioni (gioia, tristezza, rabbia, paura) e sentiment (positivo/negativo)
    nei file CSV in una directory. Conta le occorrenze di ogni emozione per commenti positivi e negativi, esegue un test
    del Chi-quadrato e riporta se c'è un'associazione significativa tra tipo di emozione e tipo di commento."""

    dfs = []
    emotions = ['joy', 'sadness', 'anger', 'fear']
    observed_frequencies = {'joy': {'positive': 0, 'negative': 0},
                           'sadness': {'positive': 0, 'negative': 0},
                           'anger': {'positive': 0, 'negative': 0},
                           'fear': {'positive': 0, 'negative': 0}}

```

```

for filename in os.listdir(directory_path):
    if filename.endswith(".csv"):
        filepath = os.path.join(directory_path, filename)
        df = pd.read_csv(filepath)
        dfs.append(df)
        for emotion in emotions:
            positive_count = df[df['Emozione Predetta'].str.contains(emotion) &
                                (df['Sentiment Positivo'] > df['Sentiment Negativo'])].shape[0]
            negative_count = df[df['Emozione Predetta'].str.contains(emotion) &
                                (df['Sentiment Negativo'] > df['Sentiment Positivo'])].shape[0]
            observed_frequencies[emotion]['positive'] += positive_count
            observed_frequencies[emotion]['negative'] += negative_count

observed_values = [[observed_frequencies[emotion]['positive'] for emotion in emotions],
                   [observed_frequencies[emotion]['negative'] for emotion in emotions]]
chi2, p, _, _ = chi2_contingency(observed_values)

print("Test del Chi-quadro:")
print(f"Statistiche del test: {chi2}")
print(f"Valore p: {p}")

if p < 0.05:
    print("Ci sono evidenze di un'associazione significativa tra il tipo di emozione e il tipo di commento"
          " (positivo o negativo).")
else:
    print("Non ci sono evidenze di un'associazione significativa tra il tipo di emozione e il tipo di commento"
          " (positivo o negativo).")

```

La funzione **test\_emotion** è progettata per analizzare l'associazione tra le emozioni (gioia, tristezza, rabbia, paura) e il sentiment (positivo/negativo) all'interno dei file CSV, dove sono contenuti i metadati dei commenti delle interviste. Per poter fare ciò sono state inizializzate variabili che tengono traccia delle frequenze osservate delle emozioni in relazione ai commenti positivi e negativi. In seguito, viene eseguito un test del Chi-quadro per valutare se c'è un'associazione significativa tra il tipo di emozione e il tipo di commento. Il test del Chi-quadro fornisce statistiche e un valore p-value, questi indicano la probabilità di osservare le frequenze osservate se l'ipotesi nulla fosse vera, cioè se non esiste alcuna associazione tra emozione e tipo di commento. I risultati del test vengono stampati a schermo. Se il valore p-value è minore rispetto al livello di significatività, impostato a 0.05, viene indicato che ci sono evidenze di un'associazione significativa tra il tipo di emozione e il tipo di commento. Altrimenti, viene indicato che non ci sono evidenze di tale associazione.

```

Test del Chi-quadro:
Statistiche del test: 4306.51450693535
Valore p: 0.0
Ci sono evidenze di un'associazione significativa tra il tipo di emozione e il tipo di commento (positivo o negativo).

```

Il test del Chi-quadro ha prodotto una statistica del test di 4306.51 e un valore p-value che è praticamente 0. Ciò indica che c'è una differenza significativa tra le frequenze osservate e quelle attese; quindi, le emozioni presenti nei commenti sono associate in modo significativo al tipo di commento (positivo o negativo). In pratica esiste una relazione significativa tra le emozioni predette nei commenti e il loro sentiment (positivo o negativo).

```
# percorso della cartella che contiene i file con i metadati dei commenti scaricati, il livello di sentiment
# positivo e negativo, e l'emozione associata al commento
directory_path = r'C:\Users\39392\Desktop\progettoWebAnalytics\file_commenti_csv'

# chiamate delle funzioni per la creazione dei grafici e i vari test
plot_sentiment(directory_path)
plot_emotions(directory_path)
test_negative_sentiment(directory_path)
test_emotions(directory_path)
```

In questo blocco di codice viene definito il percorso della cartella contenente i file CSV di ogni intervista, e la chiamata di ogni funzione che ha come parametro la **directory\_path** appena definita.

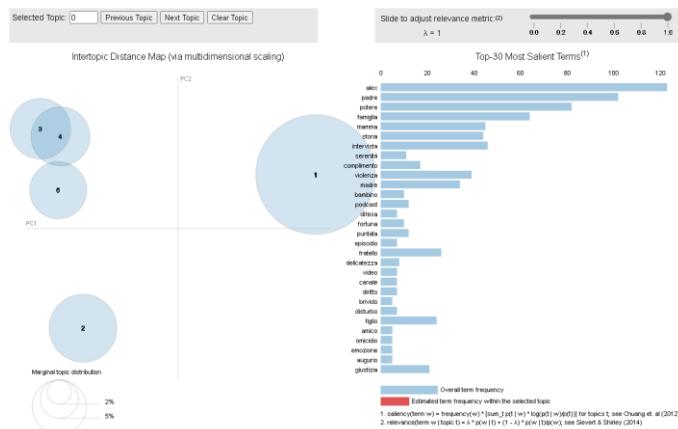
## 5) Analisi dei topic sui commenti delle interviste di One More Time su YouTube

Lo script **TopicModellingCommentsYouTube.py** esegue l'analisi dei commenti presenti nei file CSV delle interviste YouTube utilizzando il modello LDA (Latent Dirichlet Allocation) per individuare i cinque topic principali per ogni intervista. Utilizza più meno la stessa struttura sull'analisi dei topic del branch precedente, apportando solo alcune modifiche come le stopwords aggiuntive.

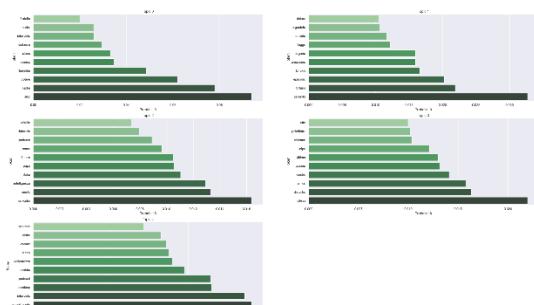
```
self.additional_stopwords = ['come', 'quando', 'sono', 'abbiamo', 'avete', 'hanno', 'sia', 'resto', 'vicenda',
'siamo', 'siete', 'essere', 'avere', 'fare', 'dire', 'detto', 'fatto', 'del',
'lha', 'lucere', 'dovere', 'totale', 'english', 'they', 'have', 'him', 'her',
'with', 'nonatola', 'aprile', 'grazie', 'dei', 'della', 'delle', 'nell', 'nella',
'nelle', 'nello', 'negli', 'come', 'about', 'conto', 'volere', 'stare', 'spessore',
'genere', 'scusa', 'mano', 'versione', 'paio', 'periodo', 'lupo', 'anch', 'gatto',
'lha', 'abbraccio', 'cuore', 'abbraccio', 'persona', 'domanda', 'risposta', 'parola',
'grazia', 'spazio', 'italiano', 'parentesi', 'ammazza', 'subtitles', 'fabrizio', 'vista',
'mammabellissima', 'will', 'niro', 'grazi', 'metà', 'sunplash', 'livello', 'parola',
'laltro', 'quando', 'alla', 'allo', 'alle', 'agli', 'senza', 'sotto', 'sopra',
'mora', 'bocca', 'resto', 'venire', 'gesto', 'situazione', 'surry', 'braccio',
'tipo', 'los', 'lodo', 'boh', 'sai', 'cioè', 'cavolo', 'lo', 'adoro', 'cerare',
'bello', 'mettere', 'totale', 'concentro', 'minuto', 'mese', 'settimana',
'sacco', 'vero', 'cosa', 'punto', 'giusto', 'super', 'perchè', 'finché', 'vabbè',
'ragazzo', 'uomo', 'donna', 'signore', 'nonantola', 'conor', 'maynard', 'boccione',
'inizio', 'fine']
```

Successivamente alla costruzione del modello LDA per ogni intervista vengono creati dei grafici e delle interfacce interattive. Tutti questi passaggi sono stati spiegati nel branch precedente. Per ogni intervista verranno inseriti le immagini che mostrano i grafici a barre accostate, grafico word clouds e l'interfaccia interattiva.

### 9. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Alex Cotoia



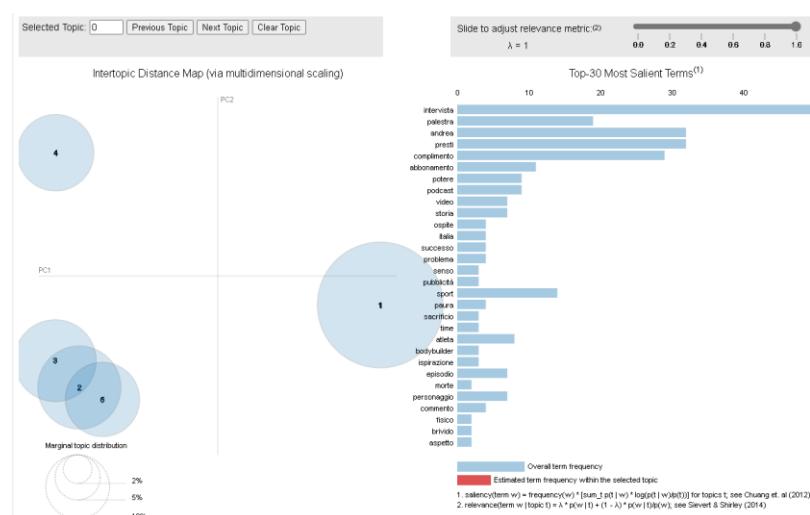
Modello di Topic - Commenti da AlexCotoia



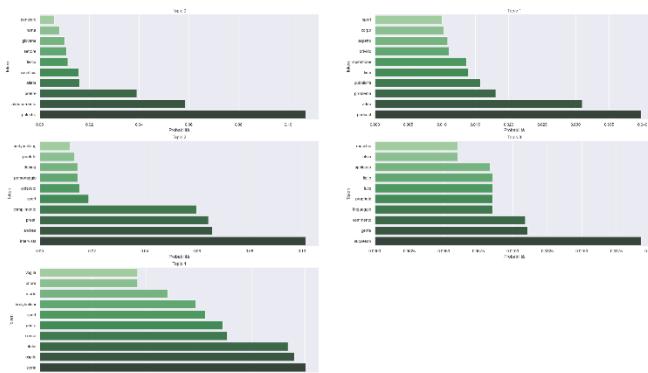
Word Clouds - Topics - AlexCotoia



## 10. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Andrea Presti



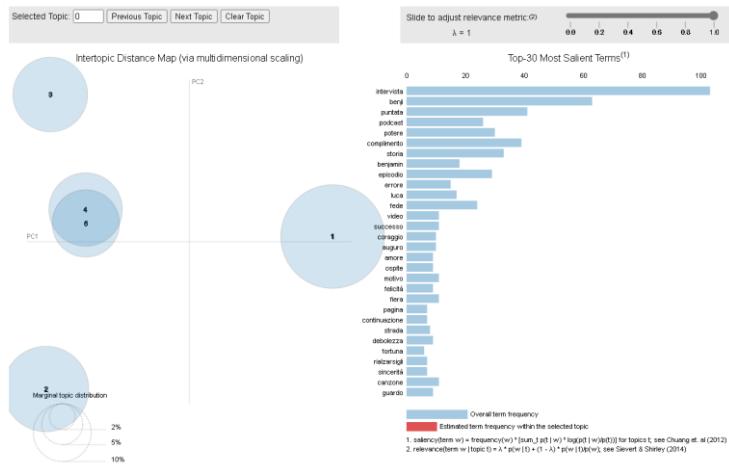
Modello di Topic - Commenti da AndreaPresti



Word Clouds - Topics - AndreaPresti

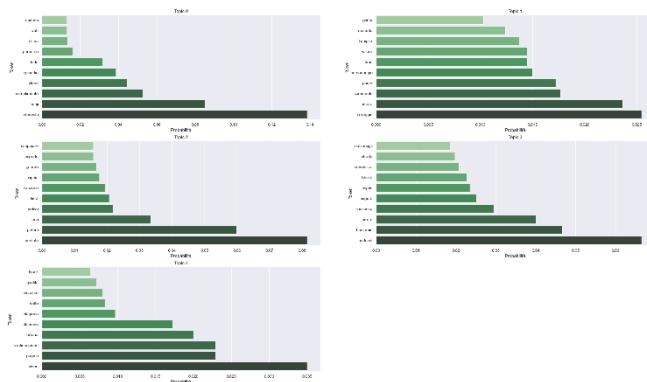


## 11. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Benjamin Mascolo

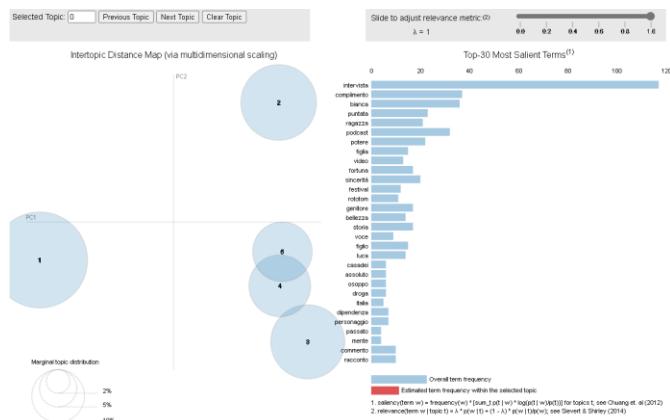


Modello di Topic - Commenti da BenjaminMascolo

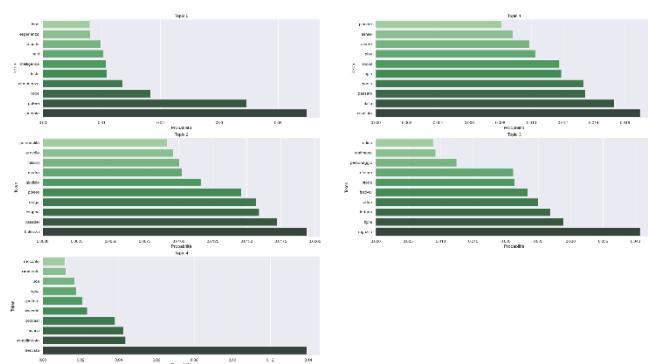
Word Clouds - Topics - BenjaminMascolo



## 12. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Bianca Balti



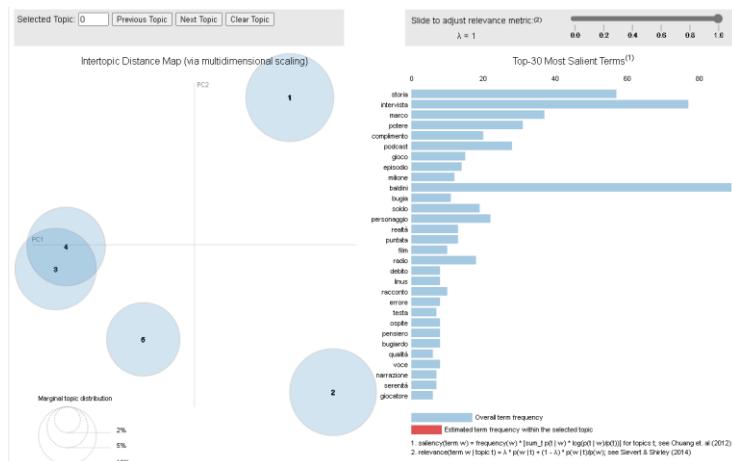
Modello di Topic - Commenti da BiancaBalti



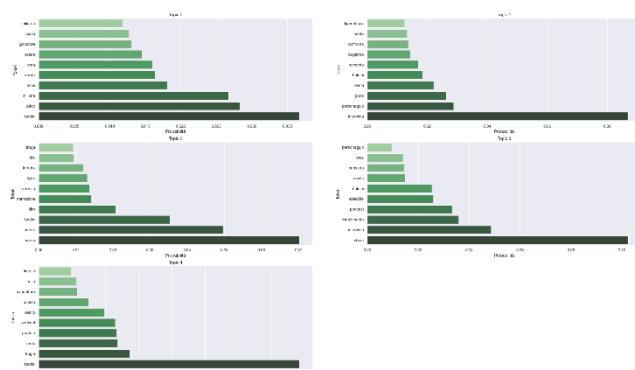
Word Clouds - Topics - BiancaBalti



### 13. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Marco Baldini



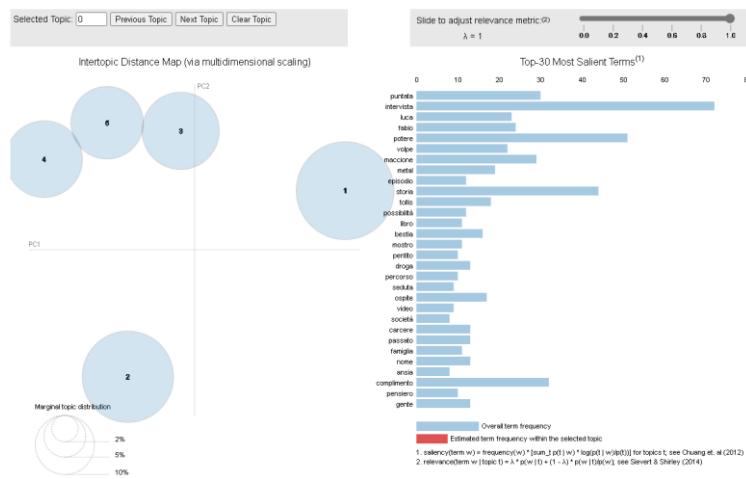
Modello di Topic - Commenti da MarcoBaldini



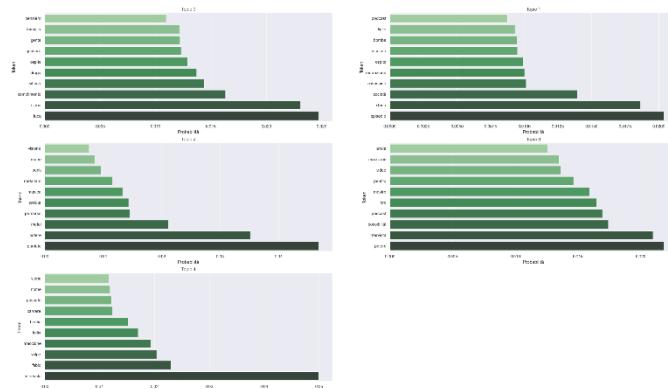
Word Clouds - Topics - MarcoBaldini



#### **14. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Mario Maccione**



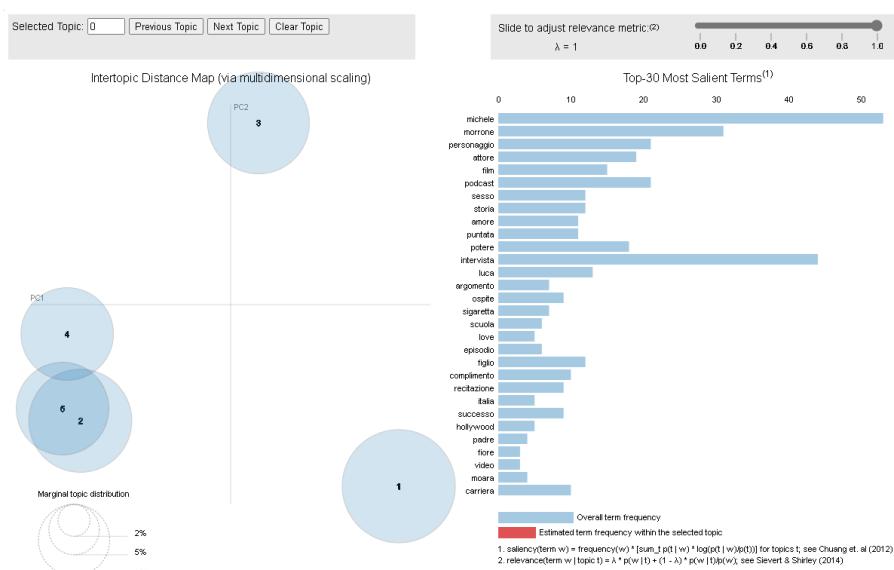
Modello di Topic - Commenti da MarioMaccione



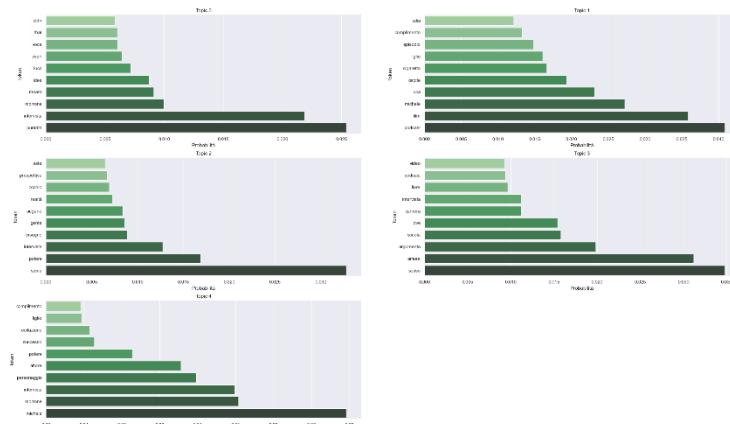
Word Clouds - Topics - MarioMaccione



*15. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Michele Morrone*



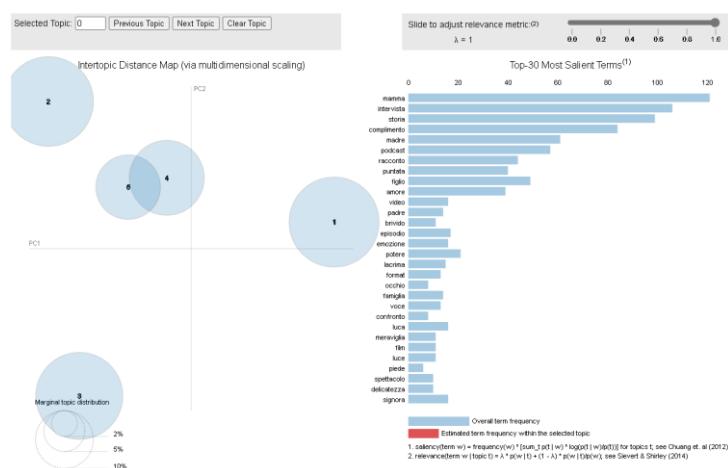
Modello di Topic - Commenti da MicheleMorrone



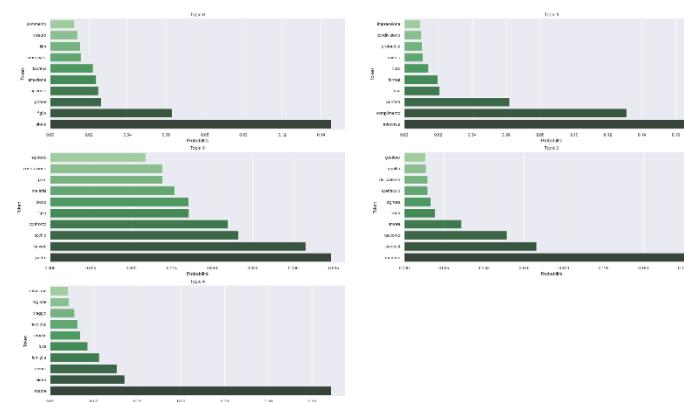
Word Clouds - Topics - MicheleMorrone



## 16. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Nicoletta Amato



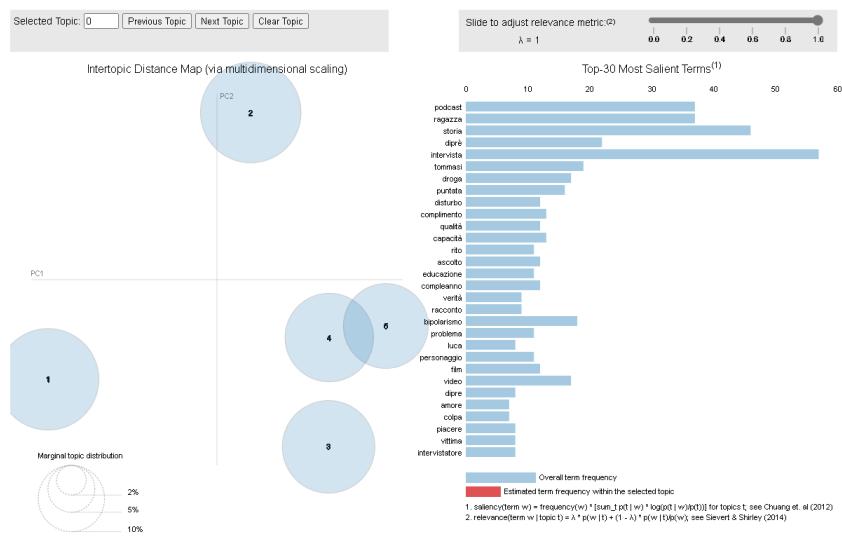
Modello di Topic - Commenti da NicolettaAmato



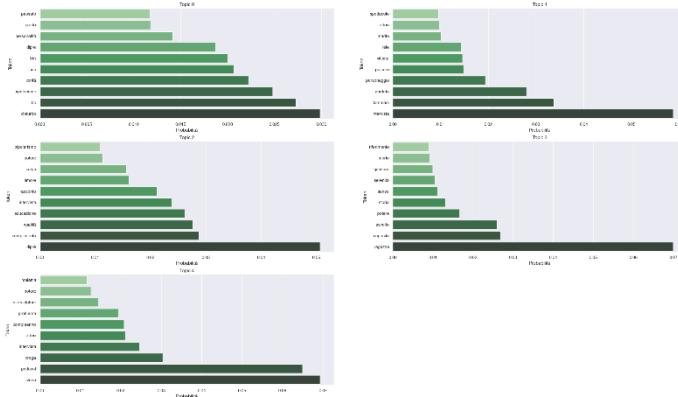
Word Clouds - Topics - NicolettaAmato



## 17. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Sara Tommasi



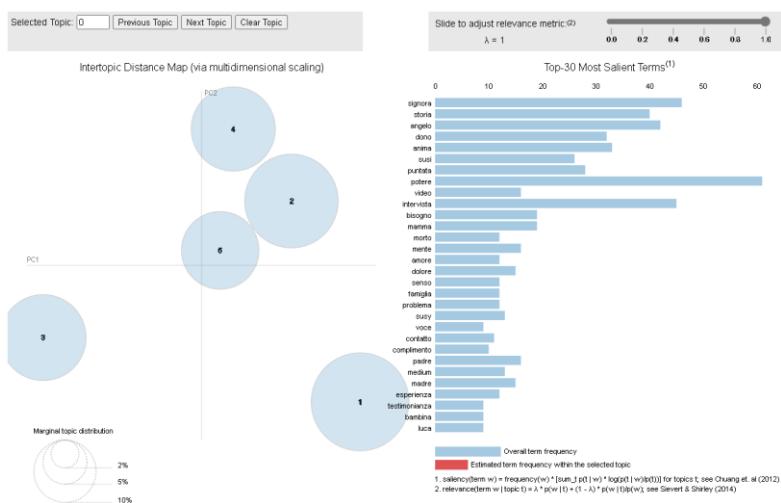
Modello di Topic - Commenti da SaraTommasi



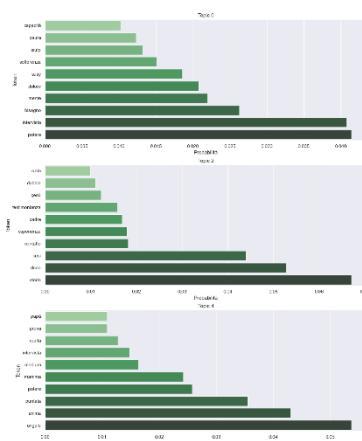
Word Clouds - Topics - SaraTommasi



## 18. Risultati immagini dei metodi appena elencati sui commenti dell'intervista a Susi Gallesi



Modello di Topic - Commenti da SusiGallesi



Word Clouds - Topics - SusiGallesi



Le parole ricorrenti come "puntata", "intervista" e "podcast" all'interno dei topic dei commenti delle interviste di "One More Time" selezionate che sono state pubblicate su YouTube descrivono chiaramente cosa accade in ogni video. Questi termini indicano che ogni video presenta un'intervista, la quale costituisce una puntata di un podcast.

Si nota dai topic che ciascuna intervista offre una struttura narrativa differente dove gli intervistati condividono le loro storie personali, esperienze e opinioni su vari temi.

- Nell'intervista di **Alex Cotoia** le parole "omicidio", "padre", "famiglia" suggeriscono che i contenuti associati trattino di storie di crimine e che ci siano collegamenti a livello familiare. Mentre ""serenità", "disturbo" e "psichiatra" indicano discussioni su salute mentale e benessere. "Difesa", "vittima" e "mostro" suggeriscono temi di crimine e difesa legale.
- Nell'intervista di **Andrea Presti** le parole come "palestra", "abbonamento", "sport", "bodybuilder", "bodybuilding" suggeriscono che l'intervista tratta come argomento in prima linea lo sport e la categoria bodybuilder. Invece le parole "paura" e "potere" potrebbero indicare delle sfide personali.
- Nell'intervista di **Benjamin Mascolo** le parole come "canzone" e "successo" possono rifletterlo il successo e l'influenza nella sua carriera musicale, mentre "amore", "coppia", "famiglia" riflettono aspetti familiari e relazionali.
- Nell'intervista di **Bianca Balti** le parole "aspetto", "bellezza", "social", "milano", "italia" possono indicare una carriera professionale basata sull'aspetto fisico nel mondo della moda. Invece, "figlia", "madre", "genitore" possono indicare legami familiari stretti.
- Nell'intervista di **Marco Baldini** le parole come "soldo", "debito" e "gioco" suggeriscono tematiche finanziarie legate al gioco d'azzardo, mentre "conduttore", "voce", "radio" possono indicare una carriera professionale all'interno di un programma radio.
- Nell'intervista di **Mario Maccione** le parole come "mostro", "assassinio", "droga" possono indicare eventi del passato che lo riguardano. Mentre "pentito" e "libro" possono indicare la pubblicazione di un'autobiografia.

- Nell'intervista di **Michele Morrone** le parole "film", "attore", "recitazione" suggeriscono una carriera nel mondo dello spettacolo, mentre le parole "amore" e "sesso" possono rilevare punti importanti nella sua vita.
- Nell'intervista di **Nicoletta Amato** i termini come "padre", "madre" e "figlio" possono indicare discussioni su forti legami familiari. Parole come "complimento" e "emozione" suggeriscono apprezzamenti ed emozioni intense, mentre "mamma" e "amore" enfatizzano le relazioni affettive e materne.
- Nell'intervista di **Sara Tommasi** parole come "disturbo" e "bipolarismo" indicano discussioni frequenti sulla salute mentale, in particolare il disturbo bipolare. Invece, "spettacolo", "personaggio", "film" possono indicare una carriera nel mondo dello spettacolo.
- Nell'intervista a **Susi Gallesi** parole ricorrenti come "angelo", "medium", "anima" suggeriscono temi spirituali ed esoterici. Si possono collegare anche alle parole "dono", "testimonianza" e "contatto".

Per ogni intervista viene valutato il livello di coerenza dei topic. Queste valutazioni vengono eseguite nello stesso modo del branch sugli audio testi.

```
# cartella contenente i file CSV delle interviste con i rispettivi commenti
input_folder = r"C:\Users\39392\Desktop\progettoWebAnalytics\file_commenti_csv"
# Cartella dove verranno salvati i risultati dell'analisi. Viene creata se non esiste già
output_folder = r"C:\Users\39392\Desktop\progettoWebAnalytics\risultati_LDA_commenti"
os.makedirs(output_folder, exist_ok=True)
# creazione di un elenco di tutti i file CSV presenti nella cartella di input
csv_files = [f for f in os.listdir(input_folder) if f.endswith('.csv')]

# lista dove viene inserito il livello di coerenza per ogni file intervista
coherence_scores = []
# passaggi da effettuare per ogni file CSV con i rispettivi commenti per eseguire l'analisi di topic
for csv_file in csv_files:
    file_path = os.path.join(input_folder, csv_file)
    topic_model = TopicModel(file_path, output_folder)
    coherence_score = topic_model.run_topic_analysis()
    coherence_scores.append({'File': csv_file, 'Coerenza': coherence_score})

# creazione di un DataFrame con i livelli di coerenza dei topic per ciascuna intervista
coherence_df = pd.DataFrame(coherence_scores)
# creazione di un file CSV dove vengono inseriti i risultati del DataFrame creato prima di questo commento
coherence_df.to_csv(os.path.join(output_folder, 'coerenza_topic.csv'), index=False)
print("Analisi completata e risultati salvati!")
```

In questo blocco di codice vengono definiti i percorsi dove si trovano la cartella contenente i file CSV delle interviste con i rispettivi commenti per ciascuna di esse e la cartella che conterrà i risultati dell'analisi dei topic. In quest'ultima verrà salvato un file contenente i titoli delle interviste con il rispettivo livello di coerenza.

```

usage
def process_and_plot_coherence_scores():
    """Funzione creata per ordinare (ordine decrescente) il livello di coerenza dei topic delle interviste,
    attraverso il percorso del file CSV creato appositamente per la coerenza, contenente le informazioni che
    determinato per ciascuna intervista un punteggio che può andare tra 0 e 1, che stabilisce coerenza tra i topic
    Estratti tramite il modello LDA. Creazione di un grafico boxplot che indica il livello di distribuzione
    dei vari livelli di coerenza"""

    output_folder = r"C:\Users\39392\Desktop\progettoWebAnalytics\risultati_LDA_commenti"
    save_image_path = r"C:\Users\39392\Desktop\progettoWebAnalytics\grafici_immagini_commentiYT"
    file_path = os.path.join(output_folder, 'coerenza_topic.csv')
    if not os.path.isfile(file_path):
        print(f"Il file '{file_path}' non esiste.")
        return

    coherence_df = pd.read_csv(file_path)
    coherence_df['File'] = coherence_df['File'].str.replace('.csv', '')
    coherence_df = coherence_df.sort_values(by='Coerenza', ascending=False)
    print("\nPunteggi di coerenza in ordine decrescente:")
    for file, coherence in zip(coherence_df['File'], coherence_df['Coerenza']):
        print(f"{file}: {coherence}")

    sns.set_style("darkgrid")
    sns.set_context(context="talk", font_scale=1.2)
    fig, ax = plt.subplots(figsize=(10, 6))
    ax = sns.boxplot(data=coherence_df['Coerenza'], orient='h', palette="Blues_d", whis=[0, 0.8])

    mean = np.mean(coherence_df['Coerenza'])
    median = np.median(coherence_df['Coerenza'])
    variance = np.var(coherence_df['Coerenza'])
    std_dev = np.std(coherence_df['Coerenza'])
    percentile_25 = np.percentile(coherence_df['Coerenza'], q=25)
    percentile_75 = np.percentile(coherence_df['Coerenza'], q=75)

    ax.set_title('Distribuzione dei punteggi del livello di coerenza\n dei topic sui commenti delle interviste',
                 fontsize=18, fontweight='bold')
    ax.set_xlabel('Livello di coerenza da 0 a 1', fontsize=14, fontweight='bold')
    ax.set_xlim(0, 1)
    ax.tick_params(axis='x', labelsize=12)

    # Aggiungi la legenda con le informazioni sui percentili e la mediana
    legend_labels = [
        'Xmin: 0',
        f'Q1: {percentile_25}',
        f'Me: {median}',
        f'Q2: {percentile_75}',
        'Xmax: 1'
    ]
    legend_text = '\n'.join(legend_labels)
    ax.legend([legend_text], loc='upper right', frameon=True, fontsize=12, framealpha=0.9)

    plt.tight_layout()
    image_file_path = os.path.join(save_image_path, 'boxplot_LivelloCoerenzaInterviste.png')
    plt.savefig(image_file_path)
    plt.close()

    print(f"Immagine salvata in: {image_file_path}")
    print(f"Media: {mean}")
    print(f"Mediana: {median}")
    print(f"25° percentile: {percentile_25}")
    print(f"75° percentile: {percentile_75}")
    print(f"Varianza: {variance}")
    print(f"Scarto quadratico medio: {std_dev}")

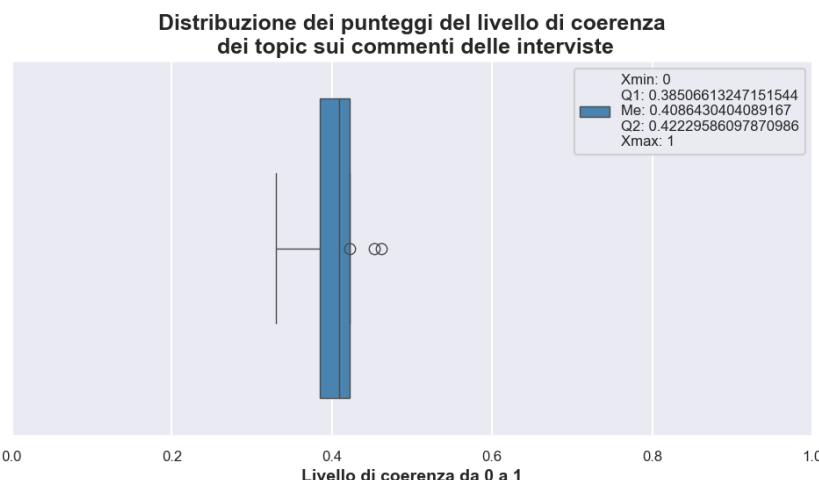
process_and_plot_coherence_scores()

```

L'ultima funzione del file script è **process\_and\_plot\_coherence\_scores()**. Legge il file CSV contenente punteggi di coerenza relativi ai topic delle interviste appena definito. Questi punteggi vengono ordinati in ordine decrescente e viene creato un box plot (l'immagine è salvata nella cartella dei risultati del modello LDA) per visualizzare la distribuzione dei punteggi di coerenza. Vengono calcolate diverse statistiche descrittive come la media, la mediana, i percentili 25° e 75°, la varianza e lo scarto quadratico medio.

Punteggi di coerenza in ordine decrescente: BiancaBalti: 0.4617551383109312 AlexCotoia: 0.4528570754143681 BenjaminMascolo: 0.422401358354336 AndreaPresti: 0.4219793688518315 MicheleMorrone: 0.4132554594958053 MarcoBaldini: 0.4040306213220281 SaraTommasi: 0.3966859152910255 NicolettaAmato: 0.3811928715316788 MarioMaccione: 0.3512003213280078 SusiGallesi: 0.3298651021602524	Media: 0.40352232320602643 Mediana: 0.4086430404089167 25° percentile: 0.38506613247151544 75° percentile: 0.42229586097870986 Varianza: 0.001532537031957979 Scarto quadratico medio: 0.03914763124325633
---	---

I livelli di coerenza dei topic, mostrati nell'output del codice, sono valori compresi da 0.33 e 0.46 circa. Ogni punteggio appartiene a una scala di valore va da 0 a 1, e indica quanto i topic estratti dal modello LDA siano coerenti all'interno di ciascuna intervista. Il punteggio di coerenza più alto, come 0.46 per Bianca Balti, indica che i topic estratti per quell'intervista sono più coerenti tra loro. Al contrario, il punteggio di coerenza più basso, come 0.33 per Susi Gallesi, suggerisce che i topic estratti per quell'intervista potrebbero essere meno coerenti. Il valore medio è di 0.4035 che è praticamente coincidente con la mediana. La varianza (approssimativamente 0.00153) indica la dispersione dei punteggi di coerenza dei topic rispetto alla loro media e avendo un valore molto basso indica che i punteggi di coerenza dei topic sono generalmente vicini alla media dei punteggi, questo suggerisce che c'è una coerenza ragionevole nella distribuzione dei punteggi di coerenza dei topic tra le interviste. Lo scarto quadratico medio (approssimativamente 0.039) rappresenta la radice quadrata della varianza e misura la dispersione dei punteggi di coerenza dei topic attorno alla media. Essendo un numero molto basso indica che i punteggi di coerenza dei topic sono, in media, vicini alla media dei punteggi, con poche deviazioni significative.



*19. Box plot che indica la distribuzione dei livelli di coerenza dei commenti delle interviste*

## **6) Conclusioni sulla seconda parte del progetto**

Come nella prima parte, l'analisi dei commenti estratti dai video delle interviste in One More Time su YouTube ha incontrato alcune difficoltà che possono essere una potenziale fonte di inattendibilità nei risultati. La lingua italiana odierna è molto contaminata sia da dialetti che da lingue straniere. Ad aggiungersi a ciò ci possono essere errori di battitura e abbreviazioni che compromettono l'accuratezza dei modelli di analisi del sentimento, delle emozioni e dei topic. Nonostante siano stati applicati preprocessamenti come rimozione di punteggiatura, conversione in minuscolo, tokenizzazione e lemmatizzazione, i modelli addestrati su testi "puliti" potrebbero non funzionare altrettanto bene su commenti che possono essere considerati come "rumorosi". Nonostante le difficoltà l'analisi sui commenti ha fornito informazioni interessanti. L'analisi del sentimento ha rivelato che, nella maggior parte delle interviste, prevalgono i commenti con sentimento positivo, ad eccezione di alcuni casi con più negatività. L'analisi sulle emozioni ha indicato la gioia come emozione predominante e la paura come l'emozione che si manifesta meno frequentemente.

Il test ANOVA ha evidenziato differenze significative nei sentimenti negativi tra le diverse interviste, mentre il test del Chi-quadrato ha confermato un'associazione significativa tra le emozioni rilevate e il sentimento positivo/negativo dei commenti. Il topic modeling applicato ha suggerito alcuni temi trattati nelle interviste che spaziano dalla carriera professionale agli aspetti familiari e personali degli intervistati. I risultati di coerenza dei topic hanno una media discreta di 0.40 (su una scala da 0 a 1) e una bassa varianza.