Tutorial API di Twitch

Nome: *Salvatore Mario*

Cognome: *Carota*

Matricola: 1000015001

Corso: SOCIAL MEDIA MANAGEMENT

Docente: *Francesco Ragusa*

Indice

Introduzione: La Piattaforma di Streaming Twitch

Twitch, la rinomata piattaforma di streaming live online, ha conquistato il cuore di milioni di spettatori e streamer in tutto il mondo, emergendo come l'epicentro dell'intrattenimento digitale in tempo reale.

Streaming di giochi

Twitch è noto principalmente per il suo streaming di giochi. Gli utenti possono trasmettere in diretta le loro sessioni di gioco, consentendo agli spettatori di seguire e interagire con il giocatore in tempo reale. Questo ha reso Twitch una destinazione popolare per gli appassionati di videogiochi che desiderano guardare e condividere esperienze di gioco.

Esports

Oltre al gaming, Twitch ospita numerosi tornei e competizioni di esports. Gli spettatori possono seguire i loro giocatori e team preferiti mentre competono in tornei di alto livello di giochi come League of Legends, Dota 2, e Counter-Strike: Global Offensive.

Diversificazione dei Contenuti di Intrattenimento

Tuttavia, Twitch va ben oltre i confini del gaming, offrendo una vasta gamma di contenuti di intrattenimento in diretta. Dagli streaming creativi alle performance musicali, dai talk show ai podcast, Twitch accoglie una varietà di interessi e passioni, unendo le menti creative di tutto il mondo sotto un unico tetto virtuale.

Unione nella Comunità Interattiva

La forza trainante di Twitch risiede nella sua vibrante comunità interattiva. La chat in tempo reale permette agli spettatori di condividere risate, emozioni e commenti con gli streamer e con gli altri spettatori, creando legami e connessioni in un ambiente digitale vivace e coinvolgente.

Sostenere i Creatori: Il Sistema di Supporto Finanziario

Infine, Twitch offre agli spettatori la possibilità di sostenere i loro creatori preferiti attraverso abbonamenti mensili, donazioni e altri gesti di supporto finanziario. Questo permette agli streamer di trasformare la loro passione in una professione, offrendo ai loro sostenitori esperienze uniche e incentivi per creare contenuti sempre più coinvolgenti e di alta qualità.

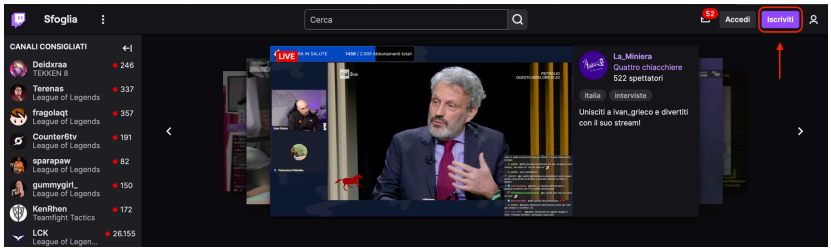
Guida all'uso delle API di Twitch

link documentazione: https://dev.twitch.tv/docs/api/

Registrazioni

Registrazione Account Twitch

Per iniziare ad usare le API di Twitch è necessario un account Twitch, per registrarsi bisogna recarsi al sito www.twitch.tv e cliccare, in alto a destra, su "**Iscriviti**" e successivamente inserire i dati richiesti.



Una volta che hai creato l'account, dovrai confermarlo cliccando sul link contenuto nell'email di verifica che ricevera.

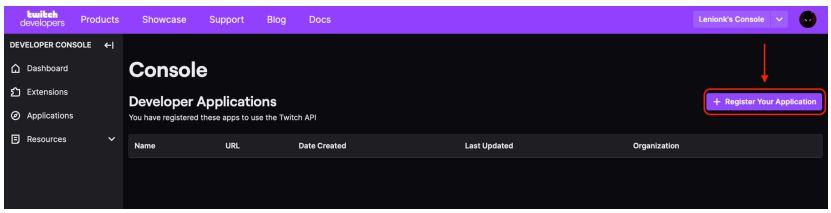
Registrazione di una Applicazione

Questa fase ci permetterà di avere accesso a delle credenziali che ci permetteranno di effettuare le chiamate al fine di svolgere analisi dei dati.
Il primo passo per ottenere un token di accesso è registrare la tua applicazione.

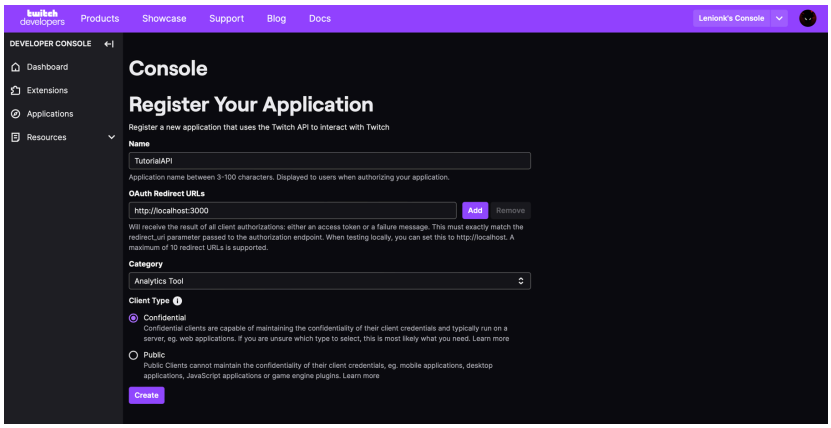
Per registrare un'applicazione:

1. Accedi alla Developer Console utilizzando il tuo account Twitch.
   Devi anche abilitare l'autenticazione a due fattori (2FA) per il tuo account. Per abilitare 2FA, vai su Sicurezza e Privacy e segui i passaggi per abilitare 2FA nella sezione Sicurezza.
   Dovrai aggiornare la console affinché queste modifiche abbiano effetto.

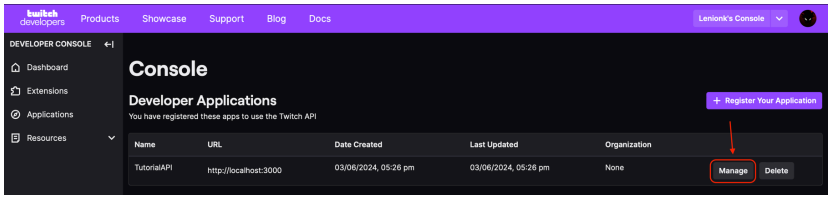
2. Seleziona la scheda "**Applications**" nella Developer Console e quindi fai clic su "**Register Your Applications**".



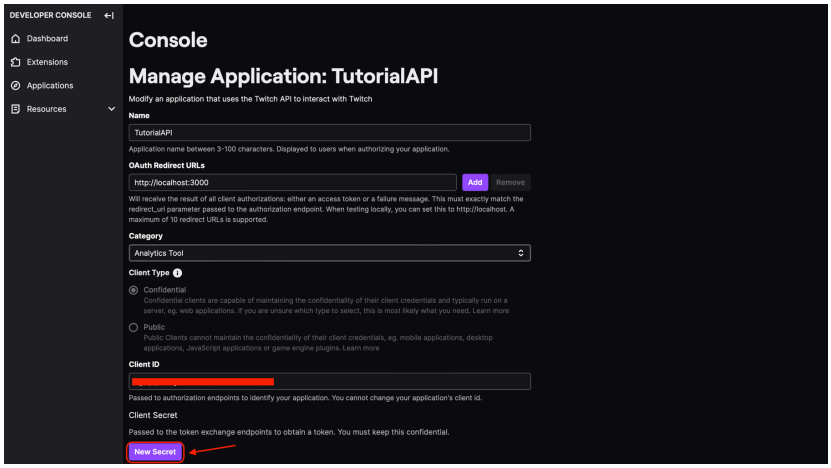
1. Imposta **Name** della tua applicazione, **Client Type** e **Category**. Il nome deve essere univoco tra tutte le applicazioni Twitch.
   Il nome della tua app è elencato nella pagina Connections sotto "**Other Connections**".



1. Torna nella scheda "**Applications**", individua la tua app in "**Developer Applications**" e fai clic su "**Menage**".



1. Conserva il "**Client ID**" e il "**Client Secret**" che otterrai dopo aver cliccato "**New Secret**"



Ottenere token di accesso OAuth

Le API di Twitch richiedono token di accesso per accedere alle risorse. A seconda della risorsa a cui accedi, avrai bisogno di un token di **accesso utente** o di **accesso all'app**.
Il **reference content** dell'API identifica il tipo di token di accesso di cui avrai bisogno. La differenza tra i due tipi di token è che un token di accesso utente ti consente di accedere ai dati sensibili di un utente (con la sua autorizzazione) e un token di accesso app ti consente di accedere solo ai suoi dati non sensibili (e non richiede l'autorizzazione dell'utente).

Per ottenere un token di accesso app sono richiesti:

Parameter	Required?	Type	Description
client_id	Yes	String	Your app's registered client ID.
client_secret	Yes	String	Your app's registered client secret.
grant_type	Yes	String	Must be set to `client_credentials`.

1. Inseriamo il nostro **Client ID** e la nostra **Secret Key** in due dariabili

In [ ]:
```
client_ID = "inserire client ID" #Client ID
secret_KEY = "inserire Secret Key" #Secret Key
```

1. Per ottenere un token di accesso, invia una richiesta HTTP POST a https://id.twitch.tv/oauth2/token.

In [ ]:
```
import requests # ci consente di effettuare richieste POST e GET
body = {
    'client_id': client_ID,
    'client_secret': secret_KEY,
    "grant_type": 'client_credentials'
}
resp = requests.post('https://id.twitch.tv/oauth2/token', body)
```

1. Se la richiesta ha esito positivo, restituisce un token di accesso in formato Json.

In [ ]:
```
import json #Per leggere dati in formato Json
token_info = resp.json()
print(token_info)

myToken = token_info['access_token']
```

```
{'access_token': 'jic71ad16oib5yn7mqebbzeg2lymo6', 'expires_in': 5669494,
'token_type': 'bearer'}
```

Uso API

Definiamo una variabile "**headers**" che ci permetterà di ottenere risposta dall'endpoint, essa deve contenere:

- Client ID
- Access Token

In [ ]:
```
# Intestazione di autorizzazione per la richiesta
headers = {
    'Client-ID': client_ID,
    'Authorization': 'Bearer ' + myToken
}
```

Il seguente link contiene la lista degli endpoint disponibili https://dev.twitch.tv/docs/api/reference/.

È importante notare che alcuni endpoint richiedono parametri aggiuntivi per completare con successo la richiesta. Assicurati di includere tutti i parametri necessari quando esegui la richiesta.

Esempi di informazioni che possiamo ottenere

Utilizzeremo la libreria **Pandas** per ottenere una struttura dati più leggibile

In [ ]:
```
import pandas as pd
from IPython.display import display
```

In [ ]:
```
# URL dell'API di Twitch per ottenere le informazioni sugli stream in dir
live_streams_url = "https://api.twitch.tv/helix/streams?first=1"

# Esempio di richiesta per ottenere le informazioni sugli stream in diret
response_live_streams = requests.get(live_streams_url, headers=headers).j
print("\nStream in diretta:")

response_live_streams_df = pd.json_normalize(response_live_streams['data'
display(response_live_streams_df)
```

```
Stream in diretta:
```

	id	user_id	user_login	user_name	game_id	game_name	type	
0	43924865963	641972806	kaicenat	KaiCenat	498592	I'm Only Sleeping	live	

- `id` : L'ID univoco dello stream.
- `user_id` : L'ID dell'utente che sta trasmettendo lo stream.
- `user_name` : Il nome dell'utente che sta trasmettendo lo stream.
- `game_id` : L'ID del gioco che viene giocato nello stream.
- `type` : Il tipo di stream (ad esempio, "live").
- `title` : Il titolo dello stream.
- `viewer_count` : Il numero di spettatori attuali dello stream.
- `started_at` : Il timestamp di quando lo stream è iniziato.
- `language` : La lingua dello stream.
- `thumbnail_url` : L'URL del thumbnail dell'anteprima dello stream.

In [ ]:
```
channel_id = response_live_streams['data'][0]['user_id'] #prendiamo l'id
print("User ID:",channel_id)

# URL dell'API di Twitch per ottenere le informazioni sugli stream in dir
streamer_url = "https://api.twitch.tv/helix/channels?broadcaster_id=" + c

# Esempio di richiesta per ottenere le informazioni sugli stream in diret
response_streamer = requests.get(streamer_url, headers=headers).json()
print("\nInformazioni streamer utilizzando come chiave ID:")

response_streamers_df = pd.json_normalize(response_streamer['data'])
display(response_streamers_df)
```

```
User ID: 641972806

Informazioni streamer utilizzando come chiave ID:
```

	broadcaster_id	broadcaster_login	broadcaster_name	broadcaster_language	game_id
0	641972806	kaicenat	KaiCenat	en	498592

- `broadcaster_id` : L'ID del broadcaster associato al canale.
- `broadcaster_login` : Il login del broadcaster associato al canale.
- `broadcaster_name` : Il nome del broadcaster associato al canale.
- `broadcaster_language` : La lingua del broadcaster.
- `game_id` : L'ID del gioco principale del canale.
- `game_name` : Il nome del gioco principale del canale.
- `title` : Il titolo del canale.
- `delay` : Il ritardo del canale.
- `tags` : I tag associati al canale.
- `content_classification_labels` : Etichette di classificazione del contenuto del canale.
- `is_branded_content` : Indica se il contenuto è pubblicitario.

In [ ]:
```
channel_name = response_live_streams['data'][0]['user_name'] #prendiamo u
print("User name:",channel_name)

# URL dell'API di Twitch per ottenere le informazioni sugli stream in dir
streamer_url = "https://api.twitch.tv/helix/users?login=" + channel_name

# Esempio di richiesta per ottenere le informazioni sugli stream in diret
response_streamer = requests.get(streamer_url, headers=headers).json()
print("\nInformazioni streamer utilizzando come chiave il nome del canale

response_streamers_df = pd.json_normalize(response_streamer['data'])
display(response_streamers_df)
```

```
User name: KaiCenat

Informazioni streamer utilizzando come chiave il nome del canale
```

	id	login	display_name	type	broadcaster_type	description	
0	641972806	kaicenat	KaiCenat		partner	Come Through & Watch These Litt STREAMS!	cdn.jtvnw.net/

- `id` : L'ID del canale Twitch.
- `login` : Il nome utente (login) associato al canale Twitch.
- `display_name` : Il nome visualizzato del canale Twitch.
- `type` : Il tipo di canale (ad esempio, "stream").
- `broadcaster_type` : Il tipo di broadcaster del canale (ad esempio, "partner", "affiliate", ecc.).
- `description` : La descrizione del canale Twitch.
- `profile_image_url` : L'URL dell'immagine del profilo del canale Twitch.
- `offline_image_url` : L'URL dell'immagine offline del canale Twitch.
- `view_count` : Il numero di visualizzazioni totali del canale Twitch.
- `created_at` : La data di creazione del canale Twitch.

In [ ]:
```
# URL dell'API di Twitch per ottenere le informazioni sulle clip
clips_url = f"https://api.twitch.tv/helix/clips?broadcaster_id={channel_i

# Esempio di richiesta per ottenere le informazioni sulle clip
response_clips = requests.get(clips_url, headers=headers).json()

# Stampa le informazioni sulle clip
print("Clip più recenti dello streamer:")

response_clips_df = pd.json_normalize(response_clips['data'])
display(response_clips_df)
```

```
Clip più recenti dello streamer:
```

	id	url	
0	VivaciousFunnyMushroomJebaited-tdU0DHqKAE2uj_N7	https://clips.twitch.tv/VivaciousFunnyMushroom...	https://(

- `id` : L'ID univoco del clip.
- `url` : L'URL del clip su Twitch.
- `embed_url` : L'URL di incorporamento della clip.
- `broadcaster_id` : L'ID del broadcaster associato alla clip.
- `broadcaster_name` : Il nome del broadcaster associato alla clip.
- `creator_id` : L'ID del creatore della clip.
- `creator_name` : Il nome del creatore della clip.
- `video_id` : L'ID del video associato alla clip.
- `game_id` : L'ID del gioco associato alla clip.
- `language` : La lingua della clip.
- `title` : Il titolo della clip.
- `view_count` : Il numero di visualizzazioni della clip.
- `created_at` : Il timestamp di quando la clip è stata creata.
- `thumbnail_url` : L'URL del thumbnail della clip.

In [ ]:
```
# URL dell'API di Twitch per ottenere le informazioni sulle Categorie più
top_games_url = "https://api.twitch.tv/helix/games/top?first=5"

# Esempio di richiesta per ottenere le informazioni sulle Categorie più p
response_top_cat = requests.get(top_games_url, headers=headers).json()
print("\n Categorie più popolari attualmente:")

response_top_cat_df = pd.json_normalize(response_top_cat['data'])
display(response_top_cat_df)
```

```
 Categorie più popolari attualmente:
```

	id	name	box_art_url	igdb_id
0	509658	Just Chatting	https://static-cdn.jtvnw.net/ttv-boxart/509658...	
1	32982	Grand Theft Auto V	https://static-cdn.jtvnw.net/ttv-boxart/32982_...	1020
2	516575	VALORANT	https://static-cdn.jtvnw.net/ttv-boxart/516575...	126459
3	21779	League of Legends	https://static-cdn.jtvnw.net/ttv-boxart/21779-...	115
4	29307	Path of Exile	https://static-cdn.jtvnw.net/ttv-boxart/29307_...	1911

- `id` : L'ID univoco del gioco.
- `name` : Il nome del gioco.
- `box_art_url` : L'URL dell'immagine di anteprima del gioco.
- `igdb_id` : L'ID del gioco nel database IGDB.

Top 5 streamer italiani

Utilizzando i parametri all'interno della chiamata all'endpoint, possiamo filtrare i dati per ottenere informazioni specifiche. Ad esempio, possiamo utilizzare questa capacità per individuare i 5 streamer italiani con il numero più alto di spettatori attuali

In [ ]:
```
# URL dell'API di Twitch per ottenere i primi 5 streamer italiani
url = "https://api.twitch.tv/helix/streams?first=5&language=it&sort=desc"

# Effettua la richiesta GET
resp = requests.get(url, headers=headers)

# Verifica se la richiesta è stata eseguita con successo
if resp.status_code == 200:
    # Stampa i dati dei streamer più seguiti
    top5itastreamer = pd.json_normalize(resp.json()['data'])
    display(top5itastreamer)
else:
    print("Errore durante la richiesta:", resp.status_code)
```

	id	user_id	user_login	user_name	game_id	game_name	type	
0	42165736712	64032771	jtaz	JTaz	509667	Food & Drink	live	P CAS
1	42158199656	269532916	xbrusche_	xBrusche_	497497	Brawl Stars	live	MA 🔥M F STA GR/
2	42165605016	72932965	darkchri99	darkchri99	21779	League of Legends	live	SS pre qu gran
3	42166082136	160489367	xiuder_	Xiuder_	33214	Fortnite	live	!CC H LOI
4	41183134263	595167998	mikeleerose	MikeLeeRose	509658	Just Chatting	live	TC GIA INL

Analisi dati

Ho deciso di condurre un'analisi dettagliata dei dati relativi a due eventi nel mondo dei videogiochi. Durante il primo evento, noto come 'Capcom Highlight', esplorerò se questo tipo di eventi porta a un aumento di spettatori sulla piattaforma Twitch. Attraverso l'analisi dei dati cercherò di capire se la presentazione di nuovi giochi da parte di Capcom suscita un interesse particolare e coinvolge attivamente gli spettatori su Twitch.

Nel secondo evento, mi concentrerò sull'esaminare il comportamento degli spettatori rispetto al primo e al secondo capitolo di 'Dragon's Dogma'. Utilizzando i dati raccolti dall'API di Twitch, condurrò un'analisi comparativa per comprendere le differenze di interesse della community nei confronti dei due capitoli del gioco. Questo mi permetterà di ottenere preziose intuizioni sull'evoluzione del franchise e sul modo in cui i giocatori hanno accolto i cambiamenti e le innovazioni introdotte nel nuovo capitolo rispetto al precedente.

Capcom Highlights: Day 2

- **Data**: Lunedì 11 marzo 2024
- **Descrizione evento**: Evento organizzato da Capcom, una delle principali aziende nel settore dei videogiochi. In questo evento vengono presentate le sue ultime uscite, anteprime di giochi in arrivo e aggiornamenti sui titoli esistenti.
- **Orario inizio**: 23:00 italiane
- **Orario fine**: 23:35 italiane
- **Obiettivo**: Analizzare l'influenza sugli spettatori
- **Durata dell'analisi**: 2 giorni

Questo evento offre un'opportunità interessante per studiare il comportamento degli spettatori durante i Capcom Highlights, giorno 2. L'analisi si concentrerà sull'influenza dell'evento sugli spettatori in italia e si estenderà su un periodo di due giorni. Inoltre, verrà utilizzato come prova del funzionamento dell'algoritmo di aquisizione dati.

Aquisizione dati

Come visto nella sezione precedente **Guida all'uso API**, otteniamo il token di accesso OAuth e definiamo la variabile headers.

In [ ]:
```
import json
import numpy as np
import requests
import datetime
import time
import pandas as pd
from IPython.display import display

client_ID = "" #Client ID
secret_KEY = "" #Secret Key

body = {
    'client_id': client_ID,
    'client_secret': secret_KEY,
    "grant_type": 'client_credentials'
}
resp = requests.post('https://id.twitch.tv/oauth2/token', body)

token_info = resp.json()
print(token_info)

myToken = token_info['access_token']

headers = {
    'Client-ID': client_ID,
    'Authorization': 'Bearer ' + myToken
}
```

Definiamo l'algoritmo per l'acquisizione dei dati usando l'API di Twitch

In [ ]:
```
import requests
import json
import datetime
import time

count = 0

while count <= 23:
    new_streams = requests.get('https://api.twitch.tv/helix/streams?type=
    new_streams_info = new_streams.json()

    new_streamdata = {
        "date": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "data": new_streams_info['data']
    }

    with open(f'stream_data_2.json', 'r+', encoding="utf8") as openfile:
        streams_info = json.load(openfile)
        streams_info['pools'].append(new_streamdata)
        openfile.seek(0)
        json.dump(streams_info, openfile)

    print("Pool n"+ str(count) +" - ["+ datetime.datetime.now().strftime(
    if(count != 23): time.sleep(3600)
    count += 1

print("Fine Giorno 1")
```

In [ ]:
```
import requests
import json
import datetime
import time


def get_data(day,path): #versione automatizzata per n giorni

    """
        Aquisisce i dati JSON relativi al numero specificato di giorni e

        Args:
            day (int): Numero di giorni per cui si desidera ottenere dati
            path (str): Percorso e nome del file JSON da creare.

        Returns:
            file json
    """

    for i in range(day):

        data = {'pools': []}
        file_name = f'{path}{i+1}.json'

        with open(file_name, 'w', encoding='utf-8') as json_file:    #cre
            json.dump(data, json_file, indent=4)

        print(f"Il file '{file_name}' è stato creato e i dati sono stati

        count = 0

        while count <= 23:
            new_streams = requests.get('https://api.twitch.tv/helix/strea
            new_streams_info = new_streams.json()

            new_streamdata = { #creo una struttura dati aggiungendo la da
                "date": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:
                "data": new_streams_info['data']
            }

            with open(file_name, 'r+', encoding="utf8") as openfile:
                streams_info = json.load(openfile)
                streams_info['pools'].append(new_streamdata)
                openfile.seek(0)
                json.dump(streams_info, openfile)

            print("Pool n"+ str(count) +" - ["+ datetime.datetime.now().s
            if(count != 23 or i != day-1): time.sleep(3600) #attendo per
            count += 1

        print("Fine Giorno ",i+1)
```

In [ ]:
```
file_name = "stream_data_capcom_"
get_data(2,file_name)
```

Definiamo una funzione che ci permette di osservere i dati ottenuti

In [ ]:
```python
import json
import pandas as pd


def get_df(path):

    """
        Carica un file JSON dal percorso specificato e restituisce un Dat

        Args:
            path (str): Percorso del file JSON.

        Returns:
            pandas.DataFrame: DataFrame contenente i dati dal file JSON.
    """

    with open(path, 'r') as file:
        data = json.load(file)
    records = []

    for pool in data['pools']:
        date = pool['date']

        for item in pool['data']:
            record = {'date': date}

            for key, value in item.items():
                record[key] = value
            records.append(record)

    df = pd.DataFrame(records)

    return df
```

Il file ottenuto dall'aquisizione dati ha le seguenti informazioni:

- 'date': Orario di esecuzione del pool.
- 'id': Identificatore univoco.
- 'user_id': ID dell'utente.
- 'user_login': Nome utente di accesso dell'utente.
- 'user_name': Nome utente dell'utente.
- 'game_id': ID del gioco.
- 'game_name': Nome del gioco.
- 'type': Tipo di stream (ad esempio, "live").
- 'title': Titolo dello stream.
- 'viewer_count': Numero di spettatori.
- 'started_at': Orario di inizio dello stream.
- 'language': Lingua dello stream.
- 'thumbnail_url': URL dell'immagine in anteprima dello stream.
- 'tag_ids': ID dei tag associati allo stream.
- 'tags': Tag associati allo stream.
- 'is_mature': Indica se il contenuto è per un pubblico maturo.

In [ ]:
```
path = "data_capcom/stream_data_capcom_1.json"
display(get_df(path).head())
```

	date	id	user_id	user_login	user_name	game_id
0	2024-03-11 00:00:00	42395142345	568747744	grenbaud	GrenBaud	509658
1	2024-03-11 00:00:00	42038026328	89600394	therealmarzaa	TheRealMarzaa	33214
2	2024-03-11 00:00:00	42394889017	53065331	dariomocciatwitch	DarioMocciaTwitch	486102336
3	2024-03-11 00:00:00	42038297496	462386663	ocwsport	ocwsport	509658
4	2024-03-11 00:00:00	42395076985	97552124	freneh	Freneh	856465807

Preprocessing dei dati

Alcune informazioni nel file JSON non sono rilevanti per l'analisi dei dati che stiamo per condurre. Pertanto, intendiamo ridurre il numero di attributi nel nostro DataFrame di Pandas.

In [ ]:
```
def preprocessing(df):

    """
    Riduce la quantità di feature nel DataFrame e converte data in format

    Args:
        df (pandas.DataFrame): Il DataFrame contenente i dati da ridurre.

    Returns:
        pandas.DataFrame: Il DataFrame ridotto con le feature selezionate
    """

    # Lista delle feature
    feature_selezionate = ['date', 'user_login', 'game_name', 'viewer_cou

    new_df = df[feature_selezionate].copy()
    new_df['date'] = pd.to_datetime(new_df['date'])

    return new_df
```

In [ ]:
```
path_1 = "data_capcom/stream_data_capcom_1.json"
path_2 = "data_capcom/stream_data_capcom_2.json"
df_g1 = preprocessing(get_df(path_1))
df_g2 = preprocessing(get_df(path_2))
df_g1.sample(5)
```

Out[ ]:

	date	user_login	game_name	viewer_count	started_at	tags
299	2024-03-11 03:00:03	ocwsport	Grand Theft Auto V	1200	2024-03-10T21:55:31Z	[Italiano, Improvvisazione, Calcio, Cabaret, S...
824	2024-03-11 08:00:08	unouidol	Pokémon Community Game	3	2024-03-11T05:31:57Z	[Italiano, English, pokemon, alerts, music, Lu...
1053	2024-03-11 10:00:09	branda_28	Minecraft	7	2024-03-11T07:57:13Z	[Italiano]
1748	2024-03-11 17:00:16	mike_slot_team	Casino	74	2024-03-11T13:59:18Z	[Italiano, casinoonline, slot, slotonline, cas...
1774	2024-03-11 18:00:17	yoshi93	EA Sports FC 24	505	2024-03-11T14:04:07Z	[Italiano]

Durante un periodo di due giorni, ho effettuato una raccolta dati tramite le API di Twitch al fine di analizzare l'attività delle streams, concentrandomi in particolare sull'evento Capcom che si è svolto durante uno dei due giorni. Lo scopo dell'analisi era comprendere i picchi di attività durante l'evento Capcom e valutare se vi fossero altre tendenze significative durante la giornata.

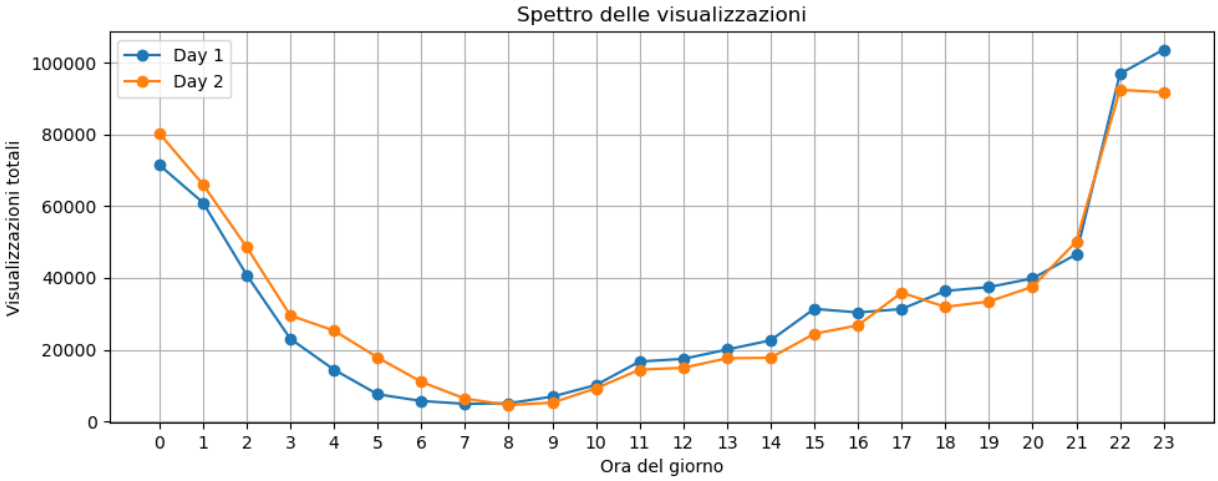
Analisi e conclusione

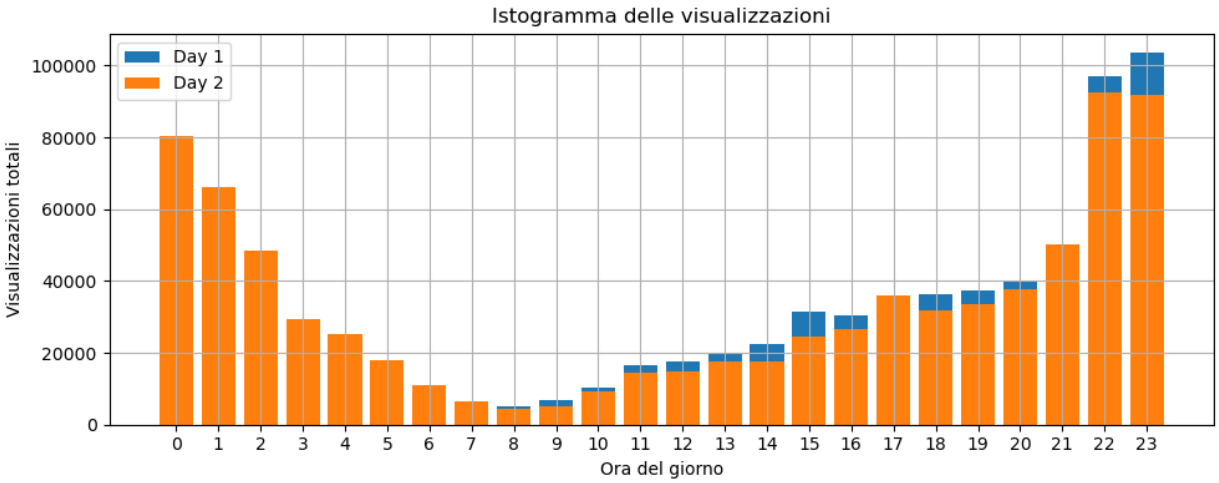
Analizziamo i dati ottenuti e usiamo la libreria matplotlib.pyplot per creare dei grafici che ci aiuterranno a comprendere meglio i dati ottenuti

In [ ]:
```
import numpy as np

def spect_ora(x,df):
    return df[df['date'].dt.hour == x]['viewer_count'].sum()
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt


Y1 = []
Y2 = []

for x in range(24):
    Y1.append(spect_ora(x, df_g1))
    Y2.append(spect_ora(x, df_g2))

fig, axs = plt.subplots(2, figsize=(10, 8))

# Primo grafico: Spettro delle visualizzazioni per le ore desiderate
axs[0].plot(range(24), Y1, marker='o', linestyle='-', label='Day 1')
axs[0].plot(range(24), Y2, marker='o', linestyle='-', label='Day 2')
axs[0].set_title('Spettro delle visualizzazioni')
axs[0].set_xlabel('Ora del giorno')
axs[0].set_ylabel('Visualizzazioni totali')
axs[0].set_xticks(range(24))
axs[0].set_xticklabels(range(24))
axs[0].legend()
axs[0].grid(True)

# Secondo grafico: Istogramma per le ore desiderate
axs[1].bar(range(24), Y1, label='Day 1')
axs[1].bar(range(24), Y2, label='Day 2')
axs[1].set_title('Istogramma delle visualizzazioni')
axs[1].set_xlabel('Ora del giorno')
axs[1].set_ylabel('Visualizzazioni totali')
axs[1].set_xticks(range(24))
axs[1].set_xticklabels(range(24))
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()
plt.show()
```





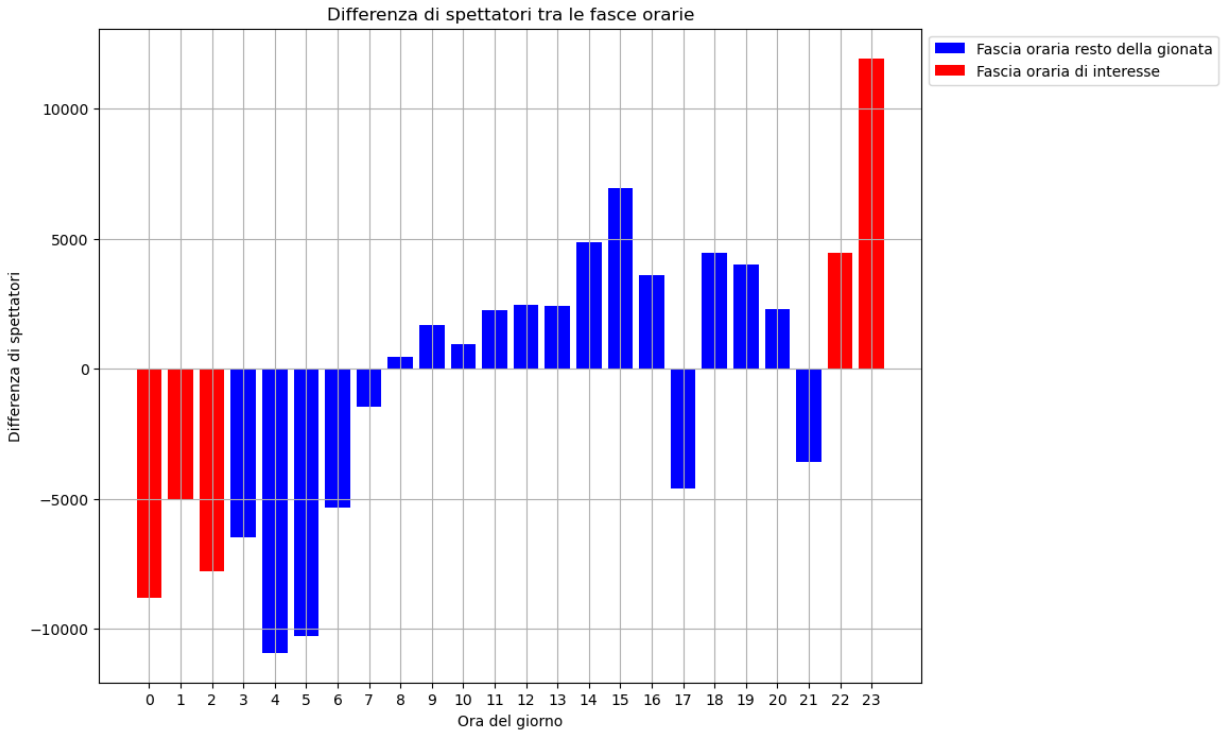
Dai grafici presentati, emerge un andamento caratteristico nello spettro delle visualizzazioni nei due giorni considerati:

- Alle ore 00:00 si osserva un significativo numero di spettatori, che poi diminuisce gradualmente fino a raggiungere le ore 7, con una notevole diminuzione.
- A partire dalle ore 8:00, si verifica una progressiva crescita della differenza di spettatori, la quale aumenta gradualmente fino alle ore 21:00.
- Successivamente, si osserva un repentino aumento dell'interesse alle ore 22:00, il quale si mantiene costante fino alle ore 23:00.

Analizziamo come la differenza di spettatori si è modificata nel corso dell'evento, focalizzandoci specificamente sulle ore 23:00 e sugli intervalli di tempo adiacenti.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Patch


# Supponiamo di avere due serie di dati Y1 e Y2 per le due fasce orarie

Y1 = []
Y2 = []

X = [22, 23, 0, 1, 2]

for ora in range(24):
    Y1.append(spect_ora(ora, df_g1))
    Y2.append(spect_ora(ora, df_g2))

# Calcolo della differenza di spettatori tra le due fasce orarie
differenza_spettatori = [(y1 - y2) for y1, y2 in zip(Y1, Y2)]

print('Valori positivi indicano un incremento di visualizzazioni nella pr
colori = ['blue' if ora not in X else 'red' for ora in range(24)]


fig, axs = plt.subplots(1, figsize=(10, 8))

# Plot della differenza di spettatori con un grafico a barre
plt.bar(range(24), differenza_spettatori, color=colori, label= 'Differenz
plt.title('Differenza di spettatori tra le fasce orarie')
plt.xlabel('Ora del giorno')
plt.ylabel('Differenza di spettatori')
plt.xticks(range(24))
plt.grid(True)

legend_elements = [Patch(facecolor='blue', label='Fascia oraria resto del
                   Patch(facecolor='red', label='Fascia oraria di interes

plt.legend(handles=legend_elements, loc='upper left', bbox_to_anchor=(1,
plt.show()
```

```
Valori positivi indicano un incremento di visualizzazioni nella prima gio
rnata
```



In [ ]:
```
differenza_spettatori[4]
```

Out[ ]:
```
-10912
```

Durante l'analisi dei dati raccolti, è emerso un pattern interessante nella differenza del numero di spettatori tra le due giornate, in particolare durante l'evento Capcom.

Il grafico mostra chiaramente un picco significativo nella differenza del numero di spettatori alle 23:00, che coincide con l'orario dell'evento. Durante questo momento, la differenza raggiunge un massimo di 11941 spettatori in più rispetto al giorno successivo, evidenziando un forte aumento dell'attività durante l'evento Capcom.

Notiamo che alle ore 4:00 e 5:00 il numero di spettatori nella seconda giornata rispetto alla prima e più alta, in particolare alle ore 4:00 la differenza è di -10912, questo suggerisce che, mentre l'evento Capcom ha generato il picco più alto di attività durante il periodo analizzato, potrebbero esserici stati altri momenti di forte interesse durante la seconda giornata, sebbene con un'intensità "leggermente" inferiore rispetto all'evento Campcom.

Questa osservazione può fornire ulteriori spunti per comprendere meglio i pattern di comportamento degli spettatori e l'effetto degli eventi sulle attività delle streams Twitch. Ad esempio sarebbe interessante analizzare più settimane per osservare se il pattern osservato nelle primi 2 grafici si ripete e associare se prensenti ai picchi di spettatori, momenti di interesse particolari legati ad eventi.

Dragon's dogma

- **Data**: 22 marzo 2024
- **Descrizione**: Videogioco di ruolo d'azione open-world sviluppato da Capcom, sequel del gioco originale.
- **Orario uscita**: 01:00 italiane
- **Obiettivo**: Analizzare tendenze di visualizzazione durante il lancio di un nuovo capitolo del gioco, e dell'interesse nel capitolo precedente.
- **Durata dell'analisi**: 10 giorni complessivi

Aquisizione dati

In [ ]:
```
import json
import numpy as np
import requests
import datetime
import time
import pandas as pd
from IPython.display import display

client_ID = "" #Client ID
secret_KEY = "" #Secret Key

body = {
    'client_id': client_ID,
    'client_secret': secret_KEY,
    "grant_type": 'client_credentials'
}
resp = requests.post('https://id.twitch.tv/oauth2/token', body)

token_info = resp.json()
print(token_info)

myToken = token_info['access_token']

headers = {
    'Client-ID': client_ID,
    'Authorization': 'Bearer ' + myToken
}
```

```
{'access_token': 'vw4vacqrvyjwtbvh6by55jm2dwlgxt', 'expires_in': 5103589,
'token_type': 'bearer'}
```

In [ ]:
```python
import requests

# Nomi dei giochi da cercare
game_name = "Dragon's Dogma II"
game_name_1 = "Dragon's Dogma: Dark Arisen"

url = f"https://api.twitch.tv/helix/games?name={game_name}"
url_1 = f"https://api.twitch.tv/helix/games?name={game_name_1}"

# Effettua la richiesta GET per ottenere gli ID dei giochi
response = requests.get(url, headers=headers)
response_1 = requests.get(url_1, headers=headers)

data = response.json()
data_1 = response_1.json()

print(data)
print(data_1)
```

```
{'data': [{'id': '435870350', 'name': "Dragon's Dogma II", 'box_art_url':
'https://static-cdn.jtvnw.net/ttv-boxart/435870350_IGDB-{width}x{height}.
jpg', 'igdb_id': '115060'}]}
{'data': [{'id': '110748', 'name': "Dragon's Dogma: Dark Arisen", 'box_ar
t_url': 'https://static-cdn.jtvnw.net/ttv-boxart/110748_IGDB-{width}x{hei
ght}.jpg', 'igdb_id': '16300'}]}
```

In [ ]:
```python
game_name = data['data'][0]['name']
game_name_1 = data_1['data'][0]['name']

game_id = data['data'][0]['id']
game_id_1 = data_1['data'][0]['id']

print(f"{game_id_1}::{game_name_1}")
print(f"{game_id}::{game_name}")
```

```
110748::Dragon's Dogma: Dark Arisen
435870350::Dragon's Dogma II
```

La differenza rispetto alla precedente acquisizione dei dati è che ora raccoglieremo i dati di due chiamate, una per ciascun gioco. Per rendere il codice più leggibile e organizzato, lo struttureremo dividendo le operazioni in funzioni.

In [ ]:
```python
import json

def init_json_file(path):

    data = {'pools': []}

    with open(path, 'w', encoding='utf-8') as json_file:    #creo file js
        json.dump(data, json_file, indent=4)

    print(f"Il file '{path}' è stato creato e i dati sono stati scritti c
```

In [ ]:
```
import json
import requests

def get_stream_data(game_id):

    url = f"https://api.twitch.tv/helix/streams?game_id={game_id}&first=1
    new_streams = requests.get(url, headers=headers)
    new_streams_info = new_streams.json()

    return {
            "date": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            "data": new_streams_info['data']
           }
```

In [ ]:
```
import json

def write_new_data_f(file_name, new_streamdata):

    with open(file_name, 'r+', encoding="utf8") as openfile:
        streams_info = json.load(openfile)
        streams_info['pools'].append(new_streamdata)
        openfile.seek(0)
        json.dump(streams_info, openfile)
```

In [ ]:
```
import requests
import json
import datetime
import time

def save_data_byGameID(game_id_1, game_id_2, day, week):

    path1 = f"data_dd/s{week}/streams_data_dd1_day{day}.json"
    path2 = f"data_dd/s{week}/streams_data_dd2_day{day}.json"

    init_json_file(path1)
    init_json_file(path2)

    count = 0

    while count <= 23:

        if(count != 0): time.sleep(3600)

        new_streamdata_1 = get_stream_data(game_id_1)
        new_streamdata_2 = get_stream_data(game_id_2)

        write_new_data_f(path1,new_streamdata_1)
        write_new_data_f(path2,new_streamdata_2)

        print(f"[Pool-{count}] - ["+ datetime.datetime.now().strftime("%Y
        count += 1

    print(f"Fine Giorno {day}")
```

In [ ]:
```
day = 5
week = 2
save_data_byGameID(game_id_1 , game_id, day, week)
```

In [ ]:
```
for day in range(5): #versione automatizzata
    if(day != 0): time.sleep(3600)
    save_data_byGameID(game_id_1 , game_id, day+1, 1)
```

Il file json ottenuto dall'aquisizione dati ha le seguenti informazioni:

- 'date': Orario di esecuzione del pool.
- 'id': Identificatore univoco.
- 'user_id': ID dell'utente.
- 'user_login': Nome utente di accesso dell'utente.
- 'user_name': Nome utente dell'utente.
- 'game_id': ID del gioco.
- 'game_name': Nome del gioco.
- 'type': Tipo di stream (ad esempio, "live").
- 'title': Titolo dello stream.
- 'viewer_count': Numero di spettatori.
- 'started_at': Orario di inizio dello stream.
- 'language': Lingua dello stream.
- 'thumbnail_url': URL dell'immagine in anteprima dello stream.
- 'tag_ids': ID dei tag associati allo stream.
- 'tags': Tag associati allo stream.
- 'is_mature': Indica se il contenuto è per un pubblico maturo.

Preprocessing dei dati

Riutiliziamo la funzone **get_df(path)** usata nella precedente analisi dati

In [ ]:
```python
import json
import pandas as pd


def get_df(path):

    """
        Carica un file JSON dal percorso specificato e restituisce un Dat

        Args:
            path (str): Percorso del file JSON.

        Returns:
            pandas.DataFrame: DataFrame contenente i dati dal file JSON.
    """

    with open(path, 'r') as file:
        data = json.load(file)
    records = []

    for pool in data['pools']:
        date = pool['date']

        for item in pool['data']:
            record = {'date': date}

            for key, value in item.items():
                record[key] = value
            records.append(record)

    df = pd.DataFrame(records)

    return df
```

In [ ]:
```python
path = "data_dd/s1/streams_data_dd1_day1.json"
df = get_df(path)
display(df.head())
```

	date	id	user_id	user_login	user_name	game_id	game
0	2024-03-15 00:00:05	43817340331	28337972	limealicious	Limealicious	110748	D Dark
1	2024-03-15 00:00:05	50631542989	608875668	zoreeeandbeans	ZoReeeAndBeans	110748	D Dark
2	2024-03-15 00:00:05	50631313005	145384084	yavanah__	Yavanah__	110748	D Dark
3	2024-03-15 00:00:05	42408370665	277040562	quaddgod	QuaddGod	110748	D Dark
4	2024-03-15 00:00:05	42407562761	406808161	beefmccat	BeefMcCat	110748	D Dark

In [ ]:
```python
def preprocessing(df):

    """
    Riduce la quantità di feature nel DataFrame e converte data in format

    Args:
        df (pandas.DataFrame): Il DataFrame contenente i dati da ridurre.

    Returns:
        pandas.DataFrame: Il DataFrame ridotto con le feature selezionate
    """

    # Lista delle feature
    feature_selezionate = ['date', 'user_login', 'game_name', 'viewer_cou

    new_df = df[feature_selezionate].copy()
    new_df['date'] = pd.to_datetime(new_df['date'])

    return new_df
```

In [ ]:
```python
path = "data_dd/s1/streams_data_dd1_day1.json"
df_d1 = preprocessing(get_df(path))
df_d1.sample(5)
```

Out[ ]:

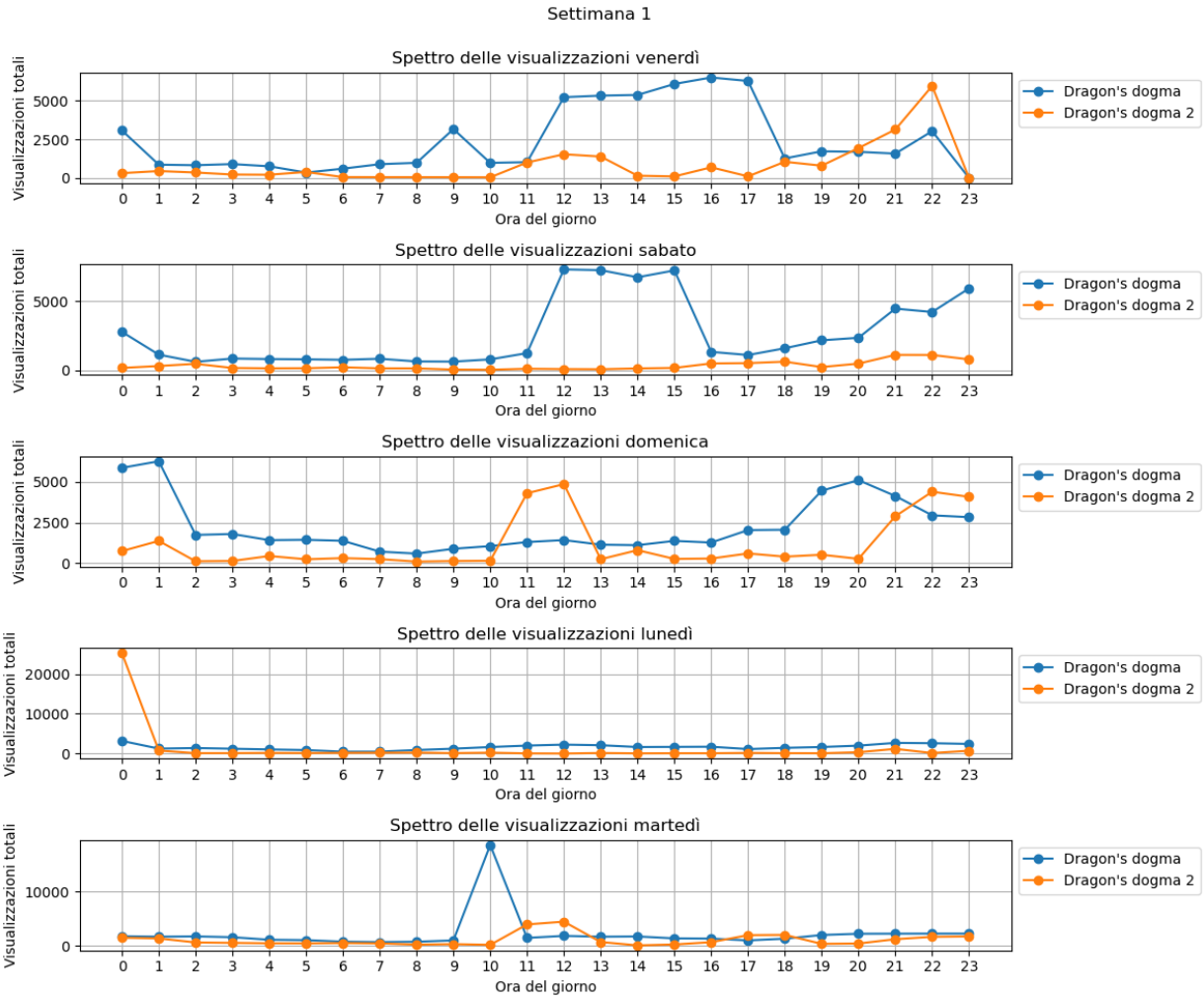
	date	user_login	game_name	viewer_count	started_at	tags
1598	2024-03-15 18:00:11	nemraka	Dragon's Dogma: Dark Arisen	1	2024-03-15T15:37:47Z	[VTuber, English, envtuber, Scottish, Chill, C...
1277	2024-03-15 15:00:07	garl____	Dragon's Dogma: Dark Arisen	2	2024-03-15T13:42:07Z	[Русский, Чилл, Прохождения]
1094	2024-03-15 13:00:04	necrotext	Dragon's Dogma: Dark Arisen	2	2024-03-15T09:04:33Z	[LGBTQIAPlus, Furry, English, Trans, Kaizo, Ch...
49	2024-03-15 00:00:05	ryn_skye	Dragon's Dogma: Dark Arisen	5	2024-03-14T22:00:16Z	[LGBTQIA, English, Bisexual, Chatty, AllWelcom...
2045	2024-03-15 22:00:18	gachapantv	Dragon's Dogma: Dark Arisen	4	2024-03-15T20:27:51Z	[vtuber, English, ENVtuber, LGBTQ, Variety]

Analisi e conclusione

Costruiamo un grafico per ogni giorno dove visualizzaremo il numero totale di spettatori per ogni fascia oraria

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

def get_prep_df(path):
    return  preprocessing(get_df(path))

def spect_ora(x,df):
    return df[df['date'].dt.hour == x]['viewer_count'].sum()

def show_bar_spect_ora(week):

    week_list = ['venerdì','sabato','domenica','lunedì','martedì']

    X = range(24)
    fig, axs = plt.subplots(5, figsize=(12, 10))

    for i in range(5):

        df = np.empty(2, dtype=object)
        for chapter in range(2):
            df[chapter] = get_prep_df(f"data_dd/s{week}/streams_data_dd{c

        Y1 = []
        Y2 = []

        for x in X:
            Y1.append(spect_ora(x,df[0]))
            Y2.append(spect_ora(x,df[1]))

        axs[i].plot(X, Y1, marker='o', linestyle='-', label="Dragon's dog
        axs[i].plot(X, Y2, marker='o', linestyle='-', label="Dragon's dog
        axs[i].set_title(f'Spettro delle visualizzazioni {week_list[i]}')
        axs[i].set_xlabel('Ora del giorno')
        axs[i].set_ylabel('Visualizzazioni totali')
        axs[i].set_xticks(X)
        axs[i].set_xticklabels(X)
        axs[i].legend(loc='upper left', bbox_to_anchor=(1, 1))
        axs[i].grid(True)



    plt.suptitle(f"Settimana {week} \n")

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
show_bar_spect_ora(1)
```



È evidente che i grafici presentano un andamento non lineare. Per comprendere meglio tali dati, approfondiremo lo studio della media.

In [ ]:
```
from tabulate import tabulate

def print_stat(chapter):
    week_list = ['Venerdì','Sabato','Domenica','Lunedì','Martedì']

    print(f"Dragon's Dogma {chapter}\n".center(80))

    for week in range(2):
        mean_week = 0

        print(f"SETTIMANA {week+1}")

        data_week = []

        for day in range(5):
            path = f"data_dd/s{week+1}/streams_data_dd{chapter}_day{day+1
            df = preprocessing(get_df(path))
            mean = df['viewer_count'].mean()
            mean_week += mean

            indice_massimo = df['viewer_count'].idxmax()
            streamer_massimo = df.loc[indice_massimo, 'user_login']

            data_week.append([week_list[day], mean, df['viewer_count'].ma

        print(tabulate(data_week, headers=['Giorno', 'Media Spettatori',

        mean_week /= 5
        print(f"Media settimana = {mean_week} \n")
```

La media visualizzata rappresenterà la media degli spettatori di ogni giorno e di ciascuno degli stream.

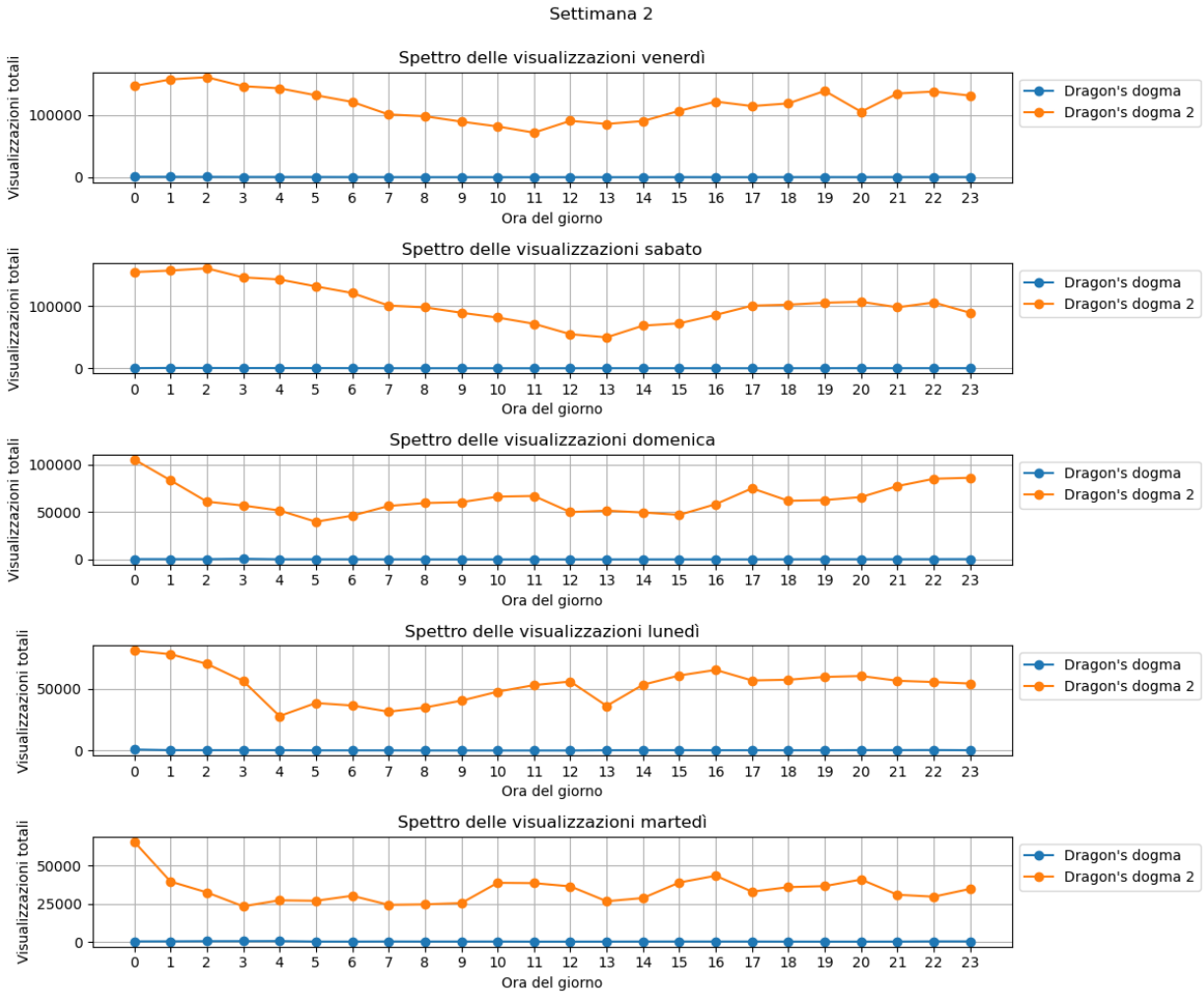
In [ ]:
```
print_stat(1)
```

```
SETTIMANA 1
+----------+--------------------+------------------+---------------------
---+
|  Giorno  |  Media Spettatori  | Picco Spettatori | Streamer con più spe
ct |
+----------+--------------------+------------------+---------------------
---+
| Venerdì  | 27.820171265461465 |       5179       |         39daph
|
|  Sabato  | 28.328558639212176 |       6325       |         39daph
|
| Domenica |  23.9118554429264  |       1941       |    holladiewaldfee
|
|  Lunedì  | 18.245786516853933 |       1305       |    holladiewaldfee
|
| Martedì  | 23.995013599274706 |      17541       |       zackrawrr
|
+----------+--------------------+------------------+---------------------
---+
Media settimana = 24.460277092745734

SETTIMANA 2
+----------+--------------------+------------------+---------------------
---+
|  Giorno  |  Media Spettatori  | Picco Spettatori | Streamer con più spe
ct |
+----------+--------------------+------------------+---------------------
---+
| Venerdì  | 4.029859484777518  |       110        |      thecalmfury
|
|  Sabato  | 3.653689715281813  |       103        |      thecalmfury
|
| Domenica | 3.3906040268456374 |       480        |      pro100funtv
|
|  Lunedì  | 4.810534591194968  |       558        |      pro100funtv
|
| Martedì  | 4.566666666666666  |       200        |         wraff
|
+----------+--------------------+------------------+---------------------
---+
Media settimana = 4.090270896953321
```

In [ ]:
```
print_stat(2)
```

```
SETTIMANA 1
+----------+--------------------+------------------+---------------------
---+
|  Giorno  |  Media Spettatori  | Picco Spettatori | Streamer con più spe
ct |
+----------+--------------------+------------------+---------------------
---+
| Venerdì  | 74.97368421052632  |       2307       |         sebjdg
|
|  Sabato  | 31.917355371900825 |       592        |         elina
|
| Domenica |  91.3993399339934  |       4774       |       melharucos
|
|  Lunedì  | 130.9145299145299  |      22990       |        cellbit
|
| Martedì  | 88.35763888888889  |       4265       |         maxim
|
+----------+--------------------+------------------+---------------------
---+
Media settimana = 83.51250966396788

SETTIMANA 2
+----------+--------------------+------------------+---------------------
---+
|  Giorno  |  Media Spettatori  | Picco Spettatori | Streamer con più spe
ct |
+----------+--------------------+------------------+---------------------
---+
| Venerdì  | 1188.792492619148  |      39596       |       zackrawrr
|
|  Sabato  | 1053.7124735729387 |      39596       |       zackrawrr
|
| Domenica | 641.6703111858704  |      24964       |       zackrawrr
|
|  Lunedì  | 537.1072182355424  |      22302       |       zackrawrr
|
| Martedì  | 342.4079831932773  |      18423       |         rubius
|
+----------+--------------------+------------------+---------------------
---+
Media settimana = 752.7380957613553
```

In [ ]:
```
show_bar_spect_ora(2)
```



Nel grafico relativo alla seconda settimana, si osserva un andamento più lineare, rendendo più chiaro il trend delle visualizzazioni.

Resoconto dell'Analisi dei Dati di Twitch per "Dragon's Dogma"

1. **Primo Capitolo:**
    - Durante la prima settimana, la media delle visualizzazioni è passata da 27 il venerdì a 23 il martedì. Questa variazione relativamente modesta potrebbe indicare una stabilità dell'interesse nel primo capitolo nel corso della settimana.
    - Nei successivi 5 giorni, la media delle visualizzazioni è diminuita a 4 spettatori, suggerendo una diminuzione dell'interesse dopo il rilascio completo del secondo capitolo.
2. **Secondo Capitolo:**
    - Durante i primi 5 giorni, la versione beta del secondo capitolo ha registrato una media di 83 visualizzazioni. Questo indica un discreto interesse nella fase di anteprima del nuovo capitolo.
    - Nei successivi 5 giorni, con il lancio completo del secondo capitolo, la media delle visualizzazioni è aumentata drasticamente a 752 spettatori, dimostrando un forte interesse dopo il rilascio completo del gioco.

Conclusioni: Nel corso della prima settimana, il primo capitolo ha mantenuto un interesse costante, ma ha subito una netta diminuzione subito dopo l'uscita del nuovo capitolo, il quale ha registrato un notevole interesse. Tuttavia, è interessante notare che il secondo capitolo ha iniziato a registrare una diminuzione delle visualizzazioni già durante la settimana del suo rilascio. Questo fenomeno potrebbe suggerire che alcuni giocatori preferiscano sperimentare il gioco personalmente anziché guardare lo stream, utilizzando la stream per valutarne il merito prima di decidere l'acquisto. Allo stesso tempo, la diminuzione delle visualizzazioni potrebbe essere attribuita a un generale calo dell'interesse nel gioco. Per approfondire ulteriormente questo fenomeno, sarebbe utile ed interessante analizzare la chat dello stream. Tuttavia, per fare ciò, è necessario essere moderatori del canale che ospita lo stream.