Progetto di Big Data

Evaluation of knowledge graph-based recommender systems

Marco Usai

60/73/65287

Capitolo 1: Introduzione

1.1 Contesto del progetto:

Nel contesto dell'evoluzione digitale, la gestione e l'analisi di volumi considerevoli di dati sono diventate fondamentali per ottenere insight significativi e prendere decisioni informate. Questo capitolo introduce il progetto, evidenziando l'importanza dell'arricchimento del dataset della musica con informazioni demografiche degli artisti.

1.2 Dataset utilizzato e obiettivo del progetto:

Il dataset preso in esame è *LFM-1b*, un dataset musicale che include un grande numero di canzoni, di artisti che le hanno create e utenti che le hanno ascoltate. Essi risiedono nei file *LFM-1b_tracks.txt, LFM-1b_users.txt, LFM-1b_artists.txt* (più altri riguardanti album e informazioni addizionali che non si sono rivelate utili a questo progetto) e le relazioni in *LFM-1b_LEs.txt*, che viene manipolato in seguito per ricavare un file che racchiude il tutto. Il problema del dataset è che non va a specificare le informazioni demografiche degli artisti. Dunque, non sappiamo effettivamente di che genere siano, quando siano nati o dove siano nati. Questo progetto mira, per l'appunto, a ricercare tali informazioni per ogni artista presente nel dataset, in modo da studiare meglio le relazioni che esistono tra gli attributi demografici e le canzoni in questione, mediante dei grafici esplicativi.

Capitolo 2: Metodologia

2.1 Utilizzo dei KG per trovare le interazioni:

L'approccio che ho impiegato è stato generare dei *knowledge graph* per individuare le relazioni che sussistessero tra utenti e tracce, per avere un quadro generale su quali sarebbero state le canzoni più ascoltate e rilevanti, in modo da filtrarle in seguito come richiesto. Un knowledge graph (grafo di conoscenza) è una struttura di dati che organizza le informazioni in modo da riflettere le relazioni esistenti tra diversi concetti o entità. In un knowledge graph, le entità sono nodi e le relazioni sono archi che collegano questi nodi. L'obiettivo principale è rappresentare in modo semantico le conoscenze in modo che sia possibile estrarre significati impliciti, scoprire nuove connessioni e facilitare la comprensione dei dati.

Per generarlo, è stato utilizzato lo script *run.py* della repository *RecSysDatasets*, che contiene degli script utili a generare dei KG per molti tipi di dataset, compreso quello utilizzato in questo progetto.

```
git clone https://github.com/RUCAIBox/RecDatasets

cd RecDatasets/conversion_tools

pip install -r requirements.txt
```

Questi sono i comandi iniziali che sono stati impiegati per clonare la repository e installare le librerie per generare il fine che contiene tutte le interazioni utente-traccia.

Precisamente, dopo aver scaricato ed estratto il dataset *LFM-1b* dal sito http://www.cp.jku.at/datasets/LFM-1b, è stato generato un file .inter di 30 GB che contiene user_id, track_id e timestamp, sostanzialmente sono tutte le tracce che sono state ascoltate da ogni utente. Ciò è stato fatto nel seguente modo:

```
python run.py --dataset lfm1b \
--input_path ../LFM-1b --output_path output_data/lfm1b \
--interaction_type tracks --convert_inter
```

2.2: Filtraggio e Aggregazione Iniziali

Gli script necessari alla riuscita del progetto richiedono delle librerie specifiche. Se si vuole riprodurre il lavoro, basterà installarle eseguendo le prime sezioni dei Notebook presenti.

Mediante lo script presente nel Notebook *filtered_extraction.ipynb*, si utilizzano Python e Pandas per eseguire un campionamento del file .inter, effettuando operazioni di filtraggio e aggregazione sui dati di interazione musicali. Si vogliono sostanzialmente le canzoni rilevanti per ogni utente, perciò sono state scelte le canzoni ascoltate collettivamente almeno 10 volte e ascoltate almeno 20 volte da un singolo utente.

Con la funzione *process_data(),* sono stati inizialmente cambiati i nomi delle colonne del file .inter, rimuovendo ":token" e ":float". Dopodiché, sono state calcolate le interazioni tra singolo utente e traccia, facendo in modo che fossero almeno 20. In seguito, è stato fatto un merge con il file contenente id e nome delle canzoni e id del corrispettivo artista, da cui si son ricavati gli ascolti collettivi, facendo sì che venissero restituite solo le canzoni con almeno 10 ascolti. Dopo aver fatto un ulteriore merge con il file che contiene tutti gli artisti, compresi i loro nomi (utili al passaggio discusso nel paragrafo seguente), è stato generato un file che contenesse tutti i risultati con le colonne elencate nell'immagine.

[3]:
```python
def process_data(interactions_file, tracks_file, artists_file, output_file):
    # Leggi i dati di interazione in un DataFrame pandas
    interactions_data = pd.read_csv(interactions_file, delimiter=',')

    # Rinomina le colonne per rimuovere i caratteri di escape
    interactions_data = interactions_data.rename(columns={
        'user_id:token': 'users_id',
        'tracks_id:token': 'tracks_id',
        'timestamp:float': 'timestamp'
    })

    # Calcola il numero di interazioni per ogni coppia utente-traccia
    user_interaction_counts = interactions_data.groupby(['users_id', 'tracks_id']).size().reset_index().rename(columns={0: 'interaction_count'})

    # Filtra le tracce ascoltate almeno 20 volte da un utente
    user_filtered_data = user_interaction_counts[user_interaction_counts['interaction_count'] >= 20]

    # Leggi i dati delle tracce in un DataFrame pandas
    tracks_data = pd.read_csv(tracks_file, sep='\t', names=['tracks_id', 'track_name', 'artist_id'])

    # Esegui il merge con i dati delle tracce
    merged_data = pd.merge(user_filtered_data, tracks_data, on='tracks_id', how='left')

    # Calcola il numero totale di ascolti per ogni traccia
    track_playcount = interactions_data.groupby('tracks_id')['tracks_id'].count().reset_index(name='total_playcount')

    # Unisci i dati filtrati con i dati totali di playcount per traccia
    final_merged_data = pd.merge(merged_data, track_playcount, on='tracks_id', how='left')

    # Leggi i dati degli artisti in un DataFrame pandas
    artists_data = pd.read_csv(artists_file, sep='\t', names=['artist_id', 'artist_name'])

    # Convert 'artist_id' column in artists_data to the same data type as in final_merged_data
    artists_data['artist_id'] = artists_data['artist_id'].astype(final_merged_data['artist_id'].dtype)

    # Esegui il merge con i dati degli artisti
    final_merged_data = pd.merge(final_merged_data, artists_data, on='artist_id', how='left')

    # Filtra le tracce ascoltate almeno 10 volte in totale
    final_filtered_data = final_merged_data[final_merged_data['total_playcount'] >= 10]

    # Seleziona le colonne desiderate
    final_columns = ['users_id', 'tracks_id', 'interaction_count', 'total_playcount', 'track_name', 'artist_name']

    # Salva l'output in un file CSV
    final_filtered_data[final_columns].to_csv(output_file, index=False)
```

3: Data augmentation su singolo nodo

3.1: Augmentation con l'ausilio di SPARQL

Lo script *augmentation_single_node.ipynb* si dedica al processo di arricchimento dei dati.

Grazie alla funzione *get_demographic_data(),* il programma esegue una query SPARQL su WikiData per andare a ricercare tutte le informazioni demografiche per ogni artista, basandosi sul suo nome. La query in questione è la seguente:

```
sparql_query = f"""
    SELECT ?artist ?genre ?birthDate ?birthPlaceLabel
    WHERE {{
       ?artist rdfs:label "{cleaned_artist_name}"@en.
       ?artist wdt:P21 ?genre;
                   wdt:P569 ?birthDate;
                   wdt:P19 ?birthPlace.

       SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
}}
    }}
    LIMIT 1
    """
```

Dove *P21, P569* e *P19* sono rispettivamente genere, data di nascita e luogo di nascita per ogni artista, il cui nome viene "ripulito" da eventuali punteggiature e simboli presenti nel dataset filtrato. La ricerca viene effettuata utilizzando come linguaggio base l'inglese, ma con "serviceParam" è possibile ottenere le etichette in varie lingue. Se l'etichetta in un certo idioma non è disponibile, verrà restituita l'etichetta in inglese come fallback. Infine, per evitare problemi di connessione, si imposta un timeout abbastanza grande di 120 secondi con 3 tentativi e si gestiscono le opportune eccezioni.

```python
# Funzione che si occupera' di effettuare la query SPARQL su Wikidata per cercare le informazioni di genere, data di nascita e luogo di nascita per ogni artista
def get_demographic_data(artist_name):
    # Controlla se artist_name è una stringa
    if not isinstance(artist_name, str):
        return {
            'genre': None,
            'birth_date': None,
            'birth_place': None
        }

    cleaned_artist_name = re.sub(r'[^\w\s&,]+', '', artist_name)

    # Gestisci il caso di nomi separati da "and" o "&"
    if '&' in cleaned_artist_name:
        cleaned_artist_name = cleaned_artist_name.split('&')[0].strip()
    elif 'and' in cleaned_artist_name:
        cleaned_artist_name = cleaned_artist_name.split('and')[0].strip()

    sparql_query = f"""
    SELECT ?artist ?genre ?birthDate ?birthPlaceLabel
    WHERE {{
      ?artist rdfs:label "{cleaned_artist_name}"@en.
      ?artist wdt:P21 ?genre;
              wdt:P569 ?birthDate;
              wdt:P19 ?birthPlace.

      SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }}
    }}
    LIMIT 1
    """
    endpoint_url = "https://query.wikidata.org/sparql"
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
        'Accept': 'application/json',
    }
    # Per evitare dei crash o dei problemi di connessione, si gestiscono le opportune eccezioni e si immette un timeout di 120 secondi.
    max_retries = 3
    for attempt in range(max_retries):
        try:
            response = requests.get(endpoint_url, headers=headers, params={'query': sparql_query, 'format': 'json'}, timeout=120)
            response.raise_for_status()
            data = response.json()
            break
        except requests.exceptions.Timeout:
            time.sleep(5)
            if attempt == max_retries - 1:
                raise
        except JSONDecodeError:
            data = None
```

Nella seconda parte dello script, di seguito riportata, vengono assegnati a ogni tupla i valori demografici trovati con la query se presenti, altrimenti si lasciano i campi vuoti.

```
    # Verifica se ci sono risultati nella lista
    if data and 'results' in data and 'bindings' in data['results'] and data['results']['bindings']:
        genre_url = data['results']['bindings'][0]['genre']['value'] if 'genre' in data['results']['bindings'][0] else None
        birth_date = data['results']['bindings'][0]['birthDate']['value'] if 'birthDate' in data['results']['bindings'][0] else None
        birth_place = data['results']['bindings'][0]['birthPlaceLabel']['value'] if 'birthPlaceLabel' in data['results']['bindings'][0] else None
        genre_id = genre_url.split('/')[-1] if genre_url else None
        genre = genre_mapping.get(genre_id, None)

        return {
            'genre': genre,
            'birth_date': birth_date,
            'birth_place': birth_place
        }
    else:
        return {
            'genre': None,
            'birth_date': None,
            'birth_place': None
        }

# Caricamento del campione iniziale di valutazioni
ratings_df = pd.read_csv("sample.csv")  # Sostituisci con il percorso del tuo file CSV

# Aggiungi colonne per i dati demografici
ratings_df['artist_genre'] = None
ratings_df['artist_birth_date'] = None
ratings_df['artist_birth_place'] = None

# Itera sul DataFrame e arricchisci con i dati demografici
for index, row in ratings_df.iterrows():
    artist_name = row['artist_name']
    demographic_data = get_demographic_data(artist_name)

    # Aggiungi i dati demografici al DataFrame
    ratings_df.at[index, 'artist_genre'] = demographic_data['genre']
    ratings_df.at[index, 'artist_birth_date'] = demographic_data['birth_date']
    ratings_df.at[index, 'artist_birth_place'] = demographic_data['birth_place']

# Salva il DataFrame arricchito
ratings_df.to_csv("user_augmentation.csv", index=False)
```

3.2: Generazione dei grafici per l'analisi dei dati

Mediante lo script *graphs_generator.ipynb*, viene preso in input il CSV prodotto dallo script precedente di augmentation, dopodiché vengono studiate le relazioni per ciascun attributo demografico. Si effettua un'analisi per valutare come le preferenze degli utenti sono distribuite tra i fornitori appartenenti a diversi gruppi demografici. Questa informazione è nota come rappresentazione di ciascun gruppo demografico (ad esempio, le artiste femminili attirano x% delle preferenze, mentre i maschi attirano y% delle preferenze).

Sostanzialmente, per ogni attributo demografico preso in considerazione, si calcola la distribuzione delle preferenze, identificate come il numero di riproduzioni delle canzoni per utente, e si generano degli areogrammi e degli istogrammi. Nel capitolo dei risultati, si vedranno più nel dettaglio.

```python
def remove_date_suffix(date_str):
    # Replace "T00:00:00Z" with an empty string
    cleaned_date = date_str.replace("T00:00:00Z", "")

    # Replace the first two characters with '19'
    cleaned_date = '19' + cleaned_date[2:] if len(cleaned_date) >= 4 else cleaned_date

    return cleaned_date

# Load the dataset, apply the converter to the relevant columns
converters = {'artist_birth_date': remove_date_suffix}
# Load the dataset
df = pd.read_csv("user_augmentation_fixed.csv", converters=converters)
# Filtra il DataFrame per includere solo 'Male' e 'Female' nell'artist_genre
filtered_df = df[df['artist_genre'].isin(['Male', 'Female'])]

# Assuming the dataset columns are 'user_id', 'track_id', 'interaction_count', 'total_playcount',
# 'track_name', 'artist_name', 'artist_genre', 'artist_birth_date', 'artist_birth_place'

# Group by demographic attributes (e.g., 'artist_genre', 'artist_birth_date', 'artist_birth_place')
demographic_groups = ['artist_genre', 'artist_birth_date', 'artist_birth_place']

for demographic_attribute in demographic_groups:
    # Group by demographic attribute
    grouped_data = filtered_df.groupby(demographic_attribute)

    # Calcola la distribuzione delle preferenze (e.g., 'total_playcount')
    distribution = grouped_data['total_playcount'].sum() / filtered_df['total_playcount'].sum() * 100

    # Mantieni solo i primi 2 valori
    distribution_pie = distribution.nlargest(2)

    # Visualizzazione: Grafico a torta
    plt.figure(figsize=(20, 10))
    if demographic_attribute == 'artist_genre':
        plt.pie(distribution_pie, labels=distribution_pie.index, startangle=45, autopct='%1.1f%%', pctdistance=0.85)
    else:
        plt.pie(distribution.nlargest(5), labels=distribution.nlargest(5).index, startangle=45, autopct='%1.1f%%', pctdistance=0.85)
    plt.title(f'Total Playcount Distribution by {demographic_attribute}')
    plt.show()

    # Visualizzazione: Grafico a barre
    plt.figure(figsize=(12, 6))
    distribution.sort_values(ascending=False).head(20).plot(kind='bar', color='skyblue')
    plt.title(f'Total Playcount Distribution by {demographic_attribute}')
    plt.xlabel(demographic_attribute)
    plt.ylabel('Percentage of Total Playcount')
    plt.xticks(rotation=45, ha='right')  # Ruota le etichette dell'asse x per una migliore leggibilità

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

Capitolo 4: Implementazione con Spark

4.1: Introduzione a Spark

Apache Spark è un framework open-source di elaborazione dati progettato per l'analisi e il processing di grandi volumi di dati in modo rapido e distribuito. Introdotto per la prima volta nel 2014, Spark ha guadagnato popolarità grazie alla sua versatilità e prestazioni superiori rispetto ad alcuni dei framework precedenti per il calcolo distribuito.

Una delle caratteristiche distintive di Apache Spark è la sua capacità di eseguire elaborazioni in memoria, consentendo di mantenere i dati in memoria RAM anziché su disco, migliorando notevolmente le prestazioni.

Apache Spark, nel suo ecosistema, include una componente chiave chiamata *DataFrame*, che estende le funzionalità di Spark per la manipolazione efficiente di dati strutturati. I DataFrame in Spark forniscono un'astrazione di alto livello simile a quelli presenti in linguaggi di programmazione come Python o R, consentendo agli utenti di eseguire operazioni SQL-like su dati distribuiti su un cluster.

4.2: Data augmentation su Spark

Il paragrafo delinea il processo di arricchimento tramite Apache Spark. Sono descritte la creazione della sessione Spark, la definizione e l'utilizzo delle User Defined Function per ottenere dati demografici degli artisti, e le operazioni di manipolazione del DataFrame risultante.

La funzione UDF utilizzata esegue la query *SPARQL* mostrata nel capitolo precedente, la struttura di output della UDF è specificata come una struttura di dati con tre campi: "artist_genre", "artist_birth_date", e "artist_birth_place".

Per svolgere il suo lavoro al meglio, si ripartiziona il file in input con la funzione *repartition()* (nel mio caso in 4 partizioni, 2 per ogni worker), e infine le varie parti vengono concatenate con uno script aggiuntivo.

Per runnare lo script, ci si assicuri che sia installata la libreria *requests* (*pip install requests*) e ovviamente la libreria . Dopo aver avviato il master e i worker nel cluster, si è effettuato il seguente comando:

./spark/bin/spark-submit --master spark://<ip-master>:<porta-master> --conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.memory=2G --conf spark.executor.instances=2 --num-executors 2 --py-files /percorso/filesystem/distribuito/augmentation_spark.py --files /percorso/filesystem/distribuito/sample.csv /percorso/filesystem/distribuito/augmentation_spark.py && python3 /percorso/filesystem/distribuito/concatenate_partitions.py

Dove lo script *concatenate_partitions.py* si occupa di concatenare i molteplici file originati dal partizionamento.

Ovviamente, il tutto è da modificare opportunamente in base al cluster utilizzato.

```
# Registra la funzione come UDF
get_demographic_data_udf = udf(get_demographic_data, StructType([
    StructField("artist_genre", StringType(), True),
    StructField("artist_birth_date", StringType(), True),
    StructField("artist_birth_place", StringType(), True)
]))

# Carica il DataFrame da un file CSV
ratings_df = spark.read.csv("/mnt/hgfs/Condivisione/sample.csv", header=True)

# Ripartiziona il DataFrame in quattro partizioni
ratings_df = ratings_df.repartition(4)

# Aggiungi colonne per i dati demografici utilizzando l'UDF
ratings_df = ratings_df.withColumn("demographic_data", get_demographic_data_udf(ratings_df["artist_name"]))

# Estrai colonne dalla struttura demographic_data
ratings_df = ratings_df \
    .withColumn("artist_genre", ratings_df["demographic_data.artist_genre"]) \
    .withColumn("artist_birth_date", ratings_df["demographic_data.artist_birth_date"]) \
    .withColumn("artist_birth_place", ratings_df["demographic_data.artist_birth_place"])

# Elimina la colonna demographic_data
ratings_df = ratings_df.drop("demographic_data")

# Salva il DataFrame arricchito
ratings_df.write.csv("/mnt/hgfs/Condivisione/user_augmentation", header=True, mode="overwrite")

# Arresta la sessione Spark
spark.stop()
```

Per generare i grafici, si potrà portare il CSV in output nella stessa directory dei Notebook e runnare *graphs_generator.ipynb,* altrimenti si potrà utilizzare lo script *graphs_generator.py* nello stesso file system in cui ci sono gli altri file condivisi, dopo aver installato la libreria Matplotlib con:

pip install matplotlib

e successivamente utilizzare il comando

python3 graphs_generator_fs.py

Capitolo 5: Risultati e analisi

5.1: Dimensioni dei campioni studiati

Il lavoro è stato effettuato su campioni di dimensione variabile, partendo da un numero di circa un milione di righe estratte dal file .inter per un singolo nodo, per poi arrivare a decine di milioni utilizzando Spark. I risultati che riporto, sono prodotti utilizzando dati delle seguenti quantità: 1M, 10M e 20M.

5.2: Output dello script filtered_extraction.ipynb

Delimiter: ,

	users_id	tracks_id	interaction_count	total_playcount	track_name	artist_name
1	1036286	2540	26	776	Porcelain	Moby
2	1036286	4946	45	910	Uprising	Muse
3	1036286	19809	34	431	Fake Empire	The National
4	1036286	20989	45	124	Heart Is a Beating Drum	The Kills
5	1036286	21009	32	496	Born This Way	Lady Gaga
6	1036286	25833	20	280	Closer	Tegan and Sara
7	1036286	25915	28	273	Madness	Muse
8	1036286	26862	21	508	Follow Me	Muse
9	1036286	27605	24	416	Inertia Creeps	Massive Attack
10	1036286	28832	20	58	Nothing But Time	Metric
11	1036286	28839	54	125	Breathing Underwater	Metric
12	1036286	28842	23	87	Artificial Nocturne	Metric
13	1036286	30370	21	78	Mea Culpa	Enigma
14	1036286	32859	42	419	Overture	Daft Punk
15	1036286	33751	41	132	Hayling	FC/Kahuna
16	1036286	35150	20	94	Halcyon + On + On	Orbital
17	1036286	52816	23	408	Next Girl	The Black Keys
18	1036286	81279	33	129	Fully Alive	Flyleaf
19	1036286	109896	35	91	Stadium Love	Metric
20	1036286	151275	24	252	Future Starts Slow	The Kills
21	1036286	151757	20	55	Lightning Bolt	Pearl Jam
22	1036286	158431	37	99	Serpents	Sharon Van Etten
23	1036286	166982	21	58	) Death (Kurayamino mix)	Rob Dougan
24	1036286	175043	43	69	Moment of Clarity	Jay-Z
25	1036286	210324	35	88	Überlin	R.E.M.
26	1036286	266514	34	34	Plästic Rädio	Viva Voce
27	1036286	302651	36	62	Down	Stone Temple Pilots
28	1036286	310698	20	54	Shambala	Beastie Boys
29	1036286	319577	21	64	Alive Alone	The Chemical Brothers
30	1036286	335821	38	81	Trying to Find a Balance	Atmosphere
31	1036286	368266	33	108	Blood For Poppies	Garbage
32	1036286	443777	40	43	The Loser Wins	Atmosphere
33	1036286	452842	33	35	ds Will Roll [Instrumental]	Yeah Yeah Yeahs
34	1036286	583932	28	39	oming Back Home to You	Atmosphere
35	1036286	584238	25	25	Been Around The World	Cracker
36	1036286	584556	20	20	Move With Me (Dub)	Neneh Cherry
37	1036286	584722	32	33	Land Of Milk And Honey	Cracker

L’output del primo script si presenta così: semplicemente, a prescindere dalla dimensione, crea un CSV contenente tutte le informazioni sopra riportate, a cui verrà applicata l’augmentation.

5.3: Output dello script augmentation_single_node.ipynb

Questo script è stato il più dispendioso in fatto di risorse e tempi di esecuzione. Giustamente, essendo eseguito su un singolo nodo, non ha usufruito dei vantaggi di Spark, e anche per un singolo milione di dati per cui è stato eseguito, i tempi sono stati molto elevati.

Ad ogni modo, l’output è stato il seguente:

Delimiter: ,

	users_id	tracks_id	interaction_count	total_playcount	track_name	artist_name	artist_genre	artist_birth_date	artist_birth_place
18147	11287273	81858	24	263	Crazy	Gnarls Barkley			
18148	11287273	84928	22	522	Feeling Good	Muse			
18149	11287273	95287	24	218	Son of a Preacher Man	Dusty Springfield	Female	1939-04-16T00:00:00Z	London
18150	11287273	112272	35	74	Them There Eyes	Billie Holiday	Female	1915-04-07T00:00:00Z	Philadelphia
18151	11287273	118990	20	48	Song For My Father	Horace Silver	Male	1928-09-02T00:00:00Z	Norwalk
18152	11287273	122371	32	38	Something	Frank Sinatra	Male	1915-12-12T00:00:00Z	Hoboken
18153	11287273	127801	70	126	Chicken-Bone Circuit	RJD2	Male	1976-05-27T00:00:00Z	Eugene
18154	11287273	129693	21	78	Woo Hoo	The 5.6.7.8's			
18155	11287273	131170	35	39	Ain't No Sunshine	Rahsaan Roland Kirk			
18156	11287273	133202	26	67	Claire	Morphine			
18157	11287273	164232	43	46	Sunny Side of the Street	Dizzy Gillespie	Male	1917-10-21T00:00:00Z	Cheraw
18158	11287273	171251	22	50	Whisky in the Jar	Thin Lizzy			
18159	11287273	178897	40	87	Circuital	My Morning Jacket			
18160	11287273	184240	20	59	That's Not Really Funny	Eels			
18161	11287273	184525	30	68	A Sunday Kind of Love	Etta James	Female	1938-01-25T00:00:00Z	Los Angeles
18162	11287273	206465	21	105	The Gunner's Dream	Pink Floyd			
18163	11287273	210289	51	54	Smooth Operator	Señor Coconut			
18164	11287273	216478	36	96	Summer Madness	Kool & The Gang			
18165	11287273	227317	29	37	Eclectic Prawn	Dumbo Gets Mad			
18166	11287273	233125	32	41	Route 66	Chuck Berry	Male	1926-10-18T00:00:00Z	St. Louis
18167	11287273	249970	26	129	Big Iron	Marty Robbins	Male	1925-09-26T00:00:00Z	Glendale
18168	11287273	265773	108	163	Blue Moon	Frank Sinatra	Male	1915-12-12T00:00:00Z	Hoboken
18169	11287273	285376	37	70	Hang on Little Tomato	Pink Martini			
18170	11287273	303074	73	84	Pale Blue Eyes	The Kills			
18171	11287273	318493	65	198	't That a Kick in the Head	Dean Martin	Male	1917-06-07T00:00:00Z	Steubenville
18172	11287273	318660	21	63	Ça plane pour moi	Plastic Bertrand			
18173	11287273	411281	54	64	Pete's Jazz	Pete Rock	Male	1970-06-21T00:00:00Z	The Bronx
18174	11287273	431423	35	82	Seem To Make You Mine	The Seeds			
18175	11287273	431845	25	26	Downtown Suzie	The Rolling Stones			
18176	11287273	499479	39	48	nfessin' (That I Love You)	Thelonious Monk	Male	1917-10-10T00:00:00Z	Rocky Mount
18177	11287273	504263	36	83	Spooky	Dusty Springfield	Female	1939-04-16T00:00:00Z	London
18178	11287273	519823	27	29	Life of a Mack	100s	Male	1992-11-26T00:00:00Z	Berkeley
18179	11287273	587462	24	27	Lisa	Willie Bobo	Male	1934-02-28T00:00:00Z	New York City

Come si può evincere dall’immagine, sono state aggiunte, per ogni tupla, tre colonne che indicano genere, data di nascita e luogo di nascita degli artisti. Grazie alla query SPARQL, i campi hanno i rispettivi valori trovati per ciascuno. I campi vuoti sono solitamente dovuti all’assenza di quegli artisti su WikiData, o alla presenza di gruppi musicali invece di singole persone. Tuttavia, gran parte dei dati sono presenti.

5.4: Output dello script augmentation_spark.ipynb e comparazione dei tempi di esecuzione

Questo script prevedeva l'utilizzo di Spark, ed è stato svolto in due modi diversi. Innanzitutto, ho provato ad eseguirlo utilizzando un solo worker, e poi nuovamente utilizzandone due, il tutto 1, 10 e 20 milioni di dati.

Un esempio di output è il seguente:

Delimiter: ,

	users_id	tracks_id	interaction_count	total_playcount	track_name	artist_name	artist_genre	artist_birth_date	artist_birth_place
18147	11287273	81858	24	263	Crazy	Gnarls Barkley			
18148	11287273	84928	22	522	Feeling Good	Muse			
18149	11287273	95287	24	218	Son of a Preacher Man	Dusty Springfield	Female	1939-04-16T00:00:00Z	London
18150	11287273	112272	35	74	Them There Eyes	Billie Holiday	Female	1915-04-07T00:00:00Z	Philadelphia
18151	11287273	118990	20	48	Song For My Father	Horace Silver	Male	1928-09-02T00:00:00Z	Norwalk
18152	11287273	122371	32	38	Something	Frank Sinatra	Male	1915-12-12T00:00:00Z	Hoboken
18153	11287273	127801	70	126	Chicken-Bone Circuit	RJD2	Male	1976-05-27T00:00:00Z	Eugene
18154	11287273	129693	21	78	Woo Hoo	The 5.6.7.8's			
18155	11287273	131170	35	39	Ain't No Sunshine	Rahsaan Roland Kirk			
18156	11287273	133202	26	67	Claire	Morphine			
18157	11287273	164232	43	46	Sunny Side of the Street	Dizzy Gillespie	Male	1917-10-21T00:00:00Z	Cheraw
18158	11287273	171251	22	50	Whisky in the Jar	Thin Lizzy			
18159	11287273	178897	40	87	Circuital	My Morning Jacket			
18160	11287273	184240	20	59	That's Not Really Funny	Eels			
18161	11287273	184525	30	68	A Sunday Kind of Love	Etta James	Female	1938-01-25T00:00:00Z	Los Angeles
18162	11287273	206465	21	105	The Gunner's Dream	Pink Floyd			
18163	11287273	210289	51	54	Smooth Operator	Señor Coconut			
18164	11287273	216478	36	96	Summer Madness	Kool & The Gang			
18165	11287273	227317	29	37	Eclectic Prawn	Dumbo Gets Mad			
18166	11287273	233125	32	41	Route 66	Chuck Berry	Male	1926-10-18T00:00:00Z	St. Louis
18167	11287273	249970	26	129	Big Iron	Marty Robbins	Male	1925-09-26T00:00:00Z	Glendale
18168	11287273	265773	108	163	Blue Moon	Frank Sinatra	Male	1915-12-12T00:00:00Z	Hoboken
18169	11287273	285376	37	70	Hang on Little Tomato	Pink Martini			
18170	11287273	303074	73	84	Pale Blue Eyes	The Kills			
18171	11287273	318493	65	198	't That a Kick in the Head	Dean Martin	Male	1917-06-07T00:00:00Z	Steubenville
18172	11287273	318660	21	63	Ça plane pour moi	Plastic Bertrand			
18173	11287273	411281	54	64	Pete's Jazz	Pete Rock	Male	1970-06-21T00:00:00Z	The Bronx
18174	11287273	431423	35	82	Seem To Make You Mine	The Seeds			
18175	11287273	431845	25	26	Downtown Suzie	The Rolling Stones			
18176	11287273	499479	39	48	nfessin' (That I Love You)	Thelonious Monk	Male	1917-10-10T00:00:00Z	Rocky Mount
18177	11287273	504263	36	83	Spooky	Dusty Springfield	Female	1939-04-16T00:00:00Z	London
18178	11287273	519823	27	29	Life of a Mack	100s	Male	1992-11-26T00:00:00Z	Berkeley
18179	11287273	587462	24	27	Lisa	Willie Bobo	Male	1934-02-28T00:00:00Z	New York City

Questo è il CSV prodotto per 10 milioni di dati, mentre i tempi di esecuzione sono i seguenti:

Un solo worker:

▾ Completed Applications (3)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20231211155503-0002	demographic_enrichment	2	2.0 GiB		2023/12/11 15:55:03	murko	FINISHED	3.9 h

Due worker, con ripartizioni:

▾ Completed Applications (12)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20231211212635-0011	demographic_enrichment	4	2.0 GiB		2023/12/11 21:26:35	murko	FINISHED	58 min

Come si può notare, è evidente che l'utilizzo di Spark con due nodi ha portato a un significativo risparmio di tempo, riducendo l'elaborazione di 3 ore per una dimensione notevole del campione, pari a 10 milioni di righe filtrate. Questo risultato è il frutto dell'approccio distribuito di Spark e della sua capacità di gestire grandi volumi di dati in parallelo su più nodi di un cluster.

Lo stesso procedimento è stato effettuato per un campione di 20 milioni di tracce.

▾ Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20231212193234-0004	demographic_enrichment	4	2.0 GiB		2023/12/12 19:32:34	murko	FINISHED	2.1 h

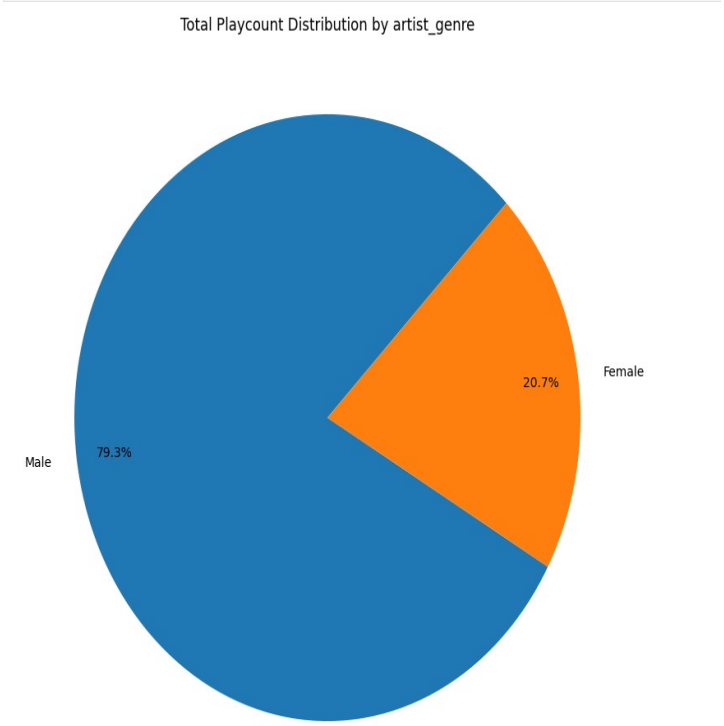
Il tempo impiegato è pari a 2 ore, per circa il doppio del contenuto. Si può evincere che anche in questo caso 2 worker hanno operato in modo più efficiente e veloce rispetto a uno solo, il quale è addirittura crashato dopo quasi 7 ore.

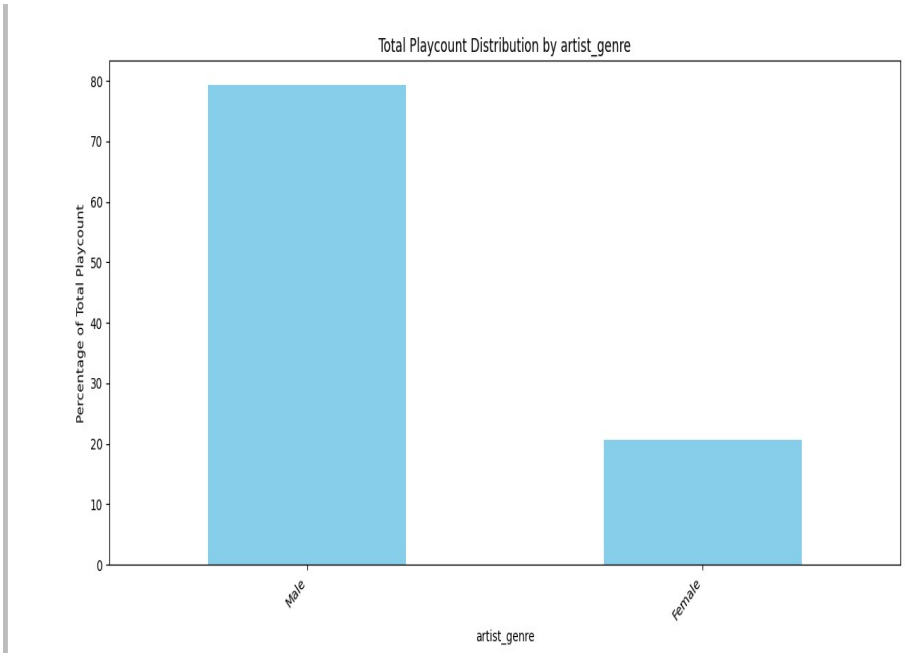
app-20231212122841-0000	demographic_enrichment	4	2.0 GiB		2023/12/12 12:28:41	murko	FINISHED	6.7 h

5.5: Analisi dei grafici

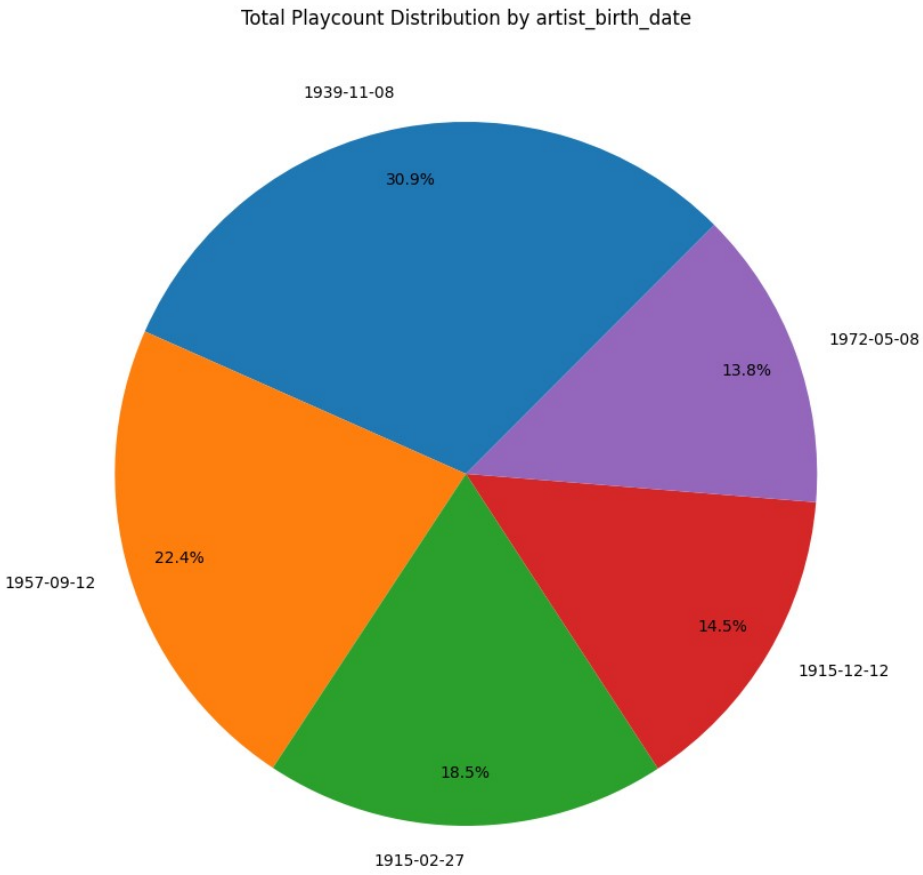
Per il campione di un milione, le relazioni instaurate tra attributo demografico e il playcount sono le seguenti:

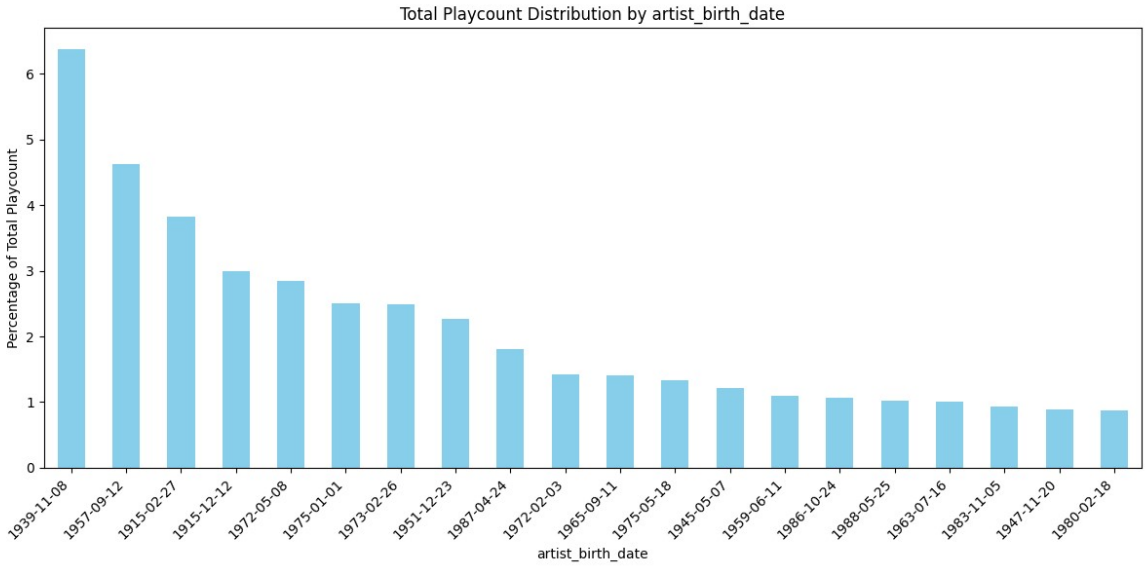
1M - Genere:



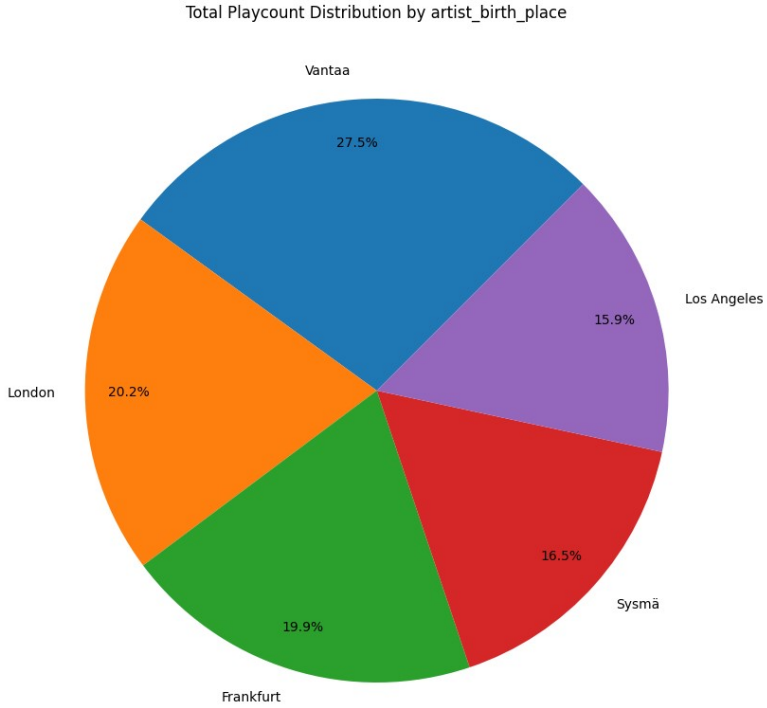


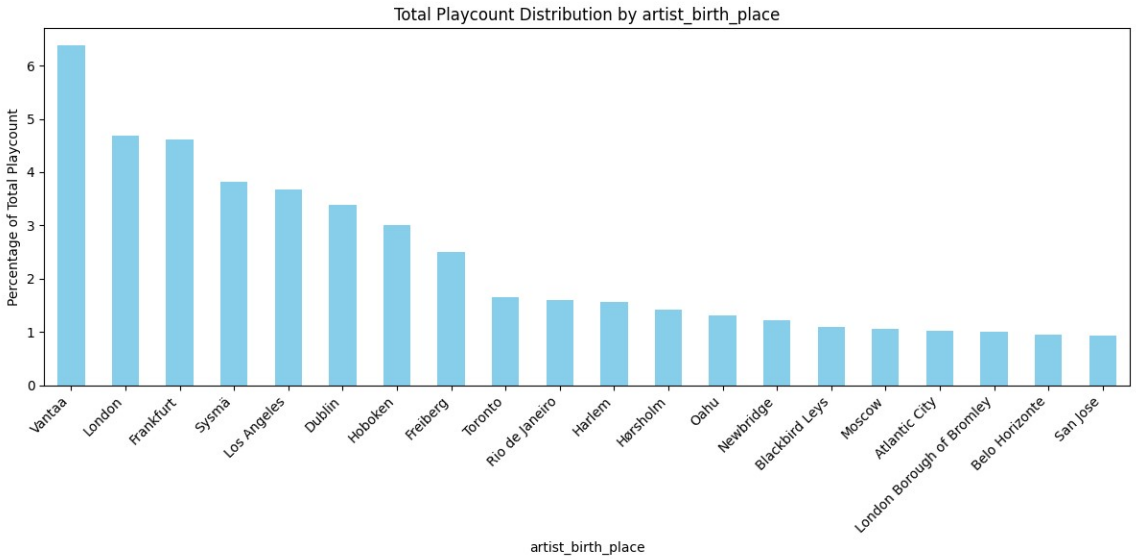
1M - Data di nascita:





1M - Luogo di nascita:

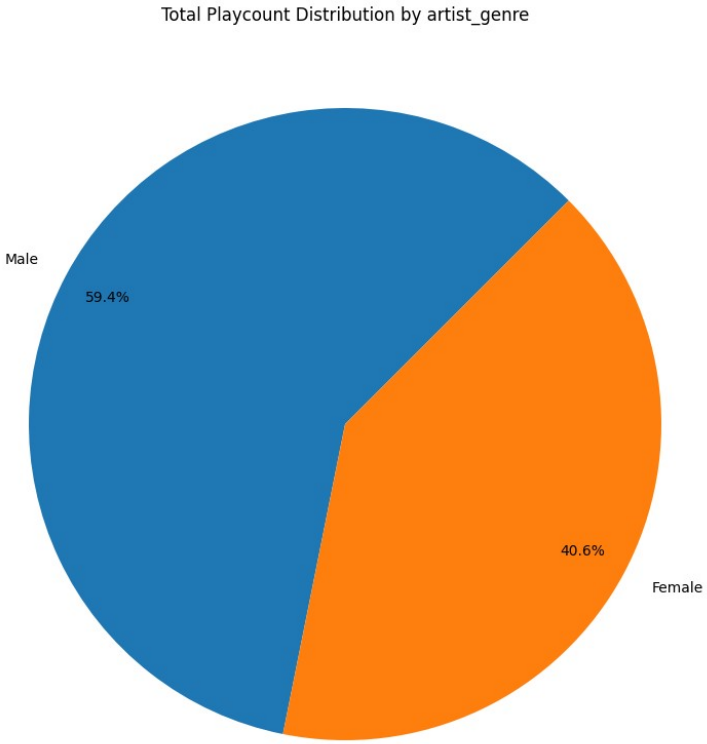


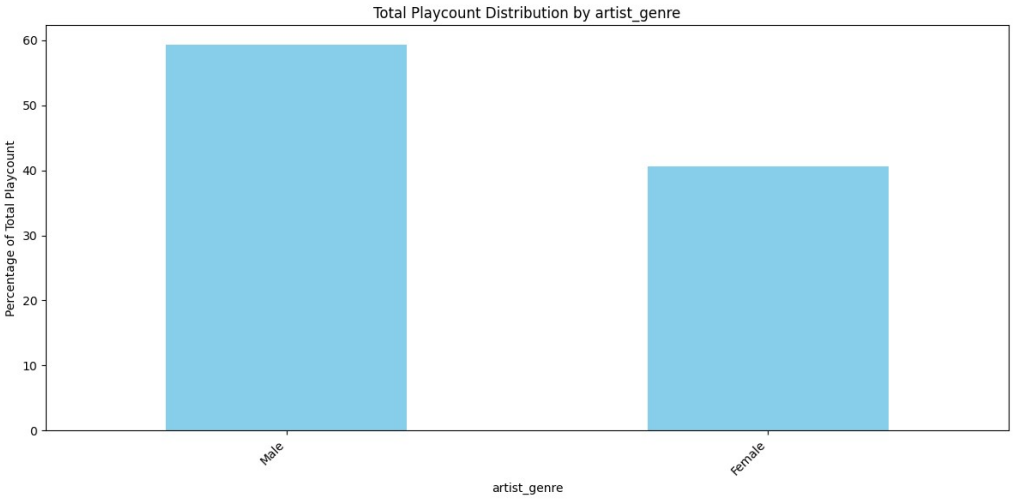


Si può evincere come una maggioranza di maschi nati a Vantaa dagli anni ‘40 agli anni ‘70 siano gli artisti più ascoltati per un campione di un milione.

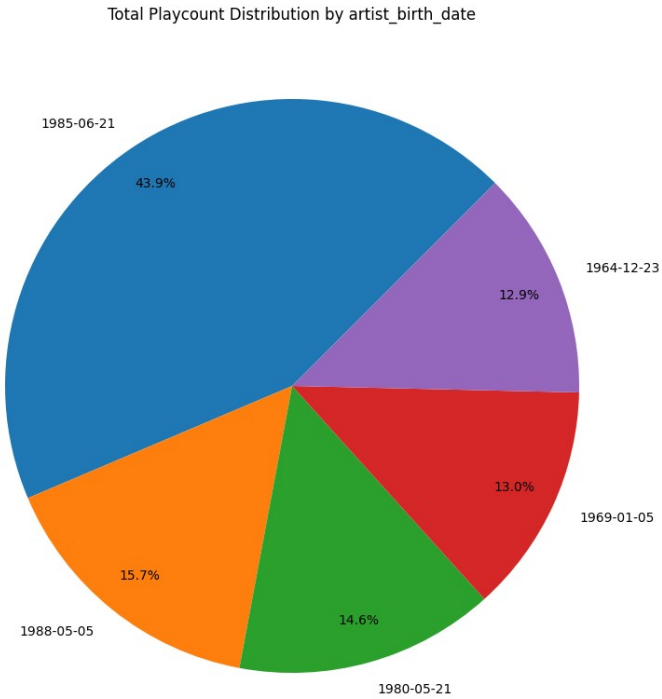
Tuttavia, questi dati non sono abbastanza coprenti, quindi tentiamo di aumentare la dimensione a 10M.

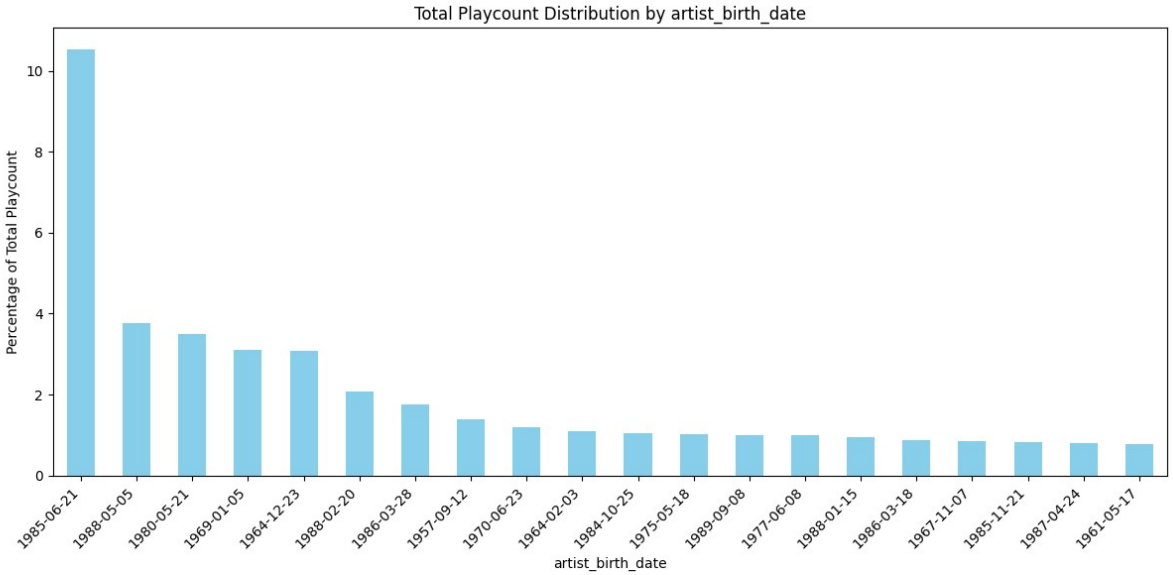
10M - Genere:


Total Playcount Distribution by artist_genre
Male
59.4%
40.6%
Female

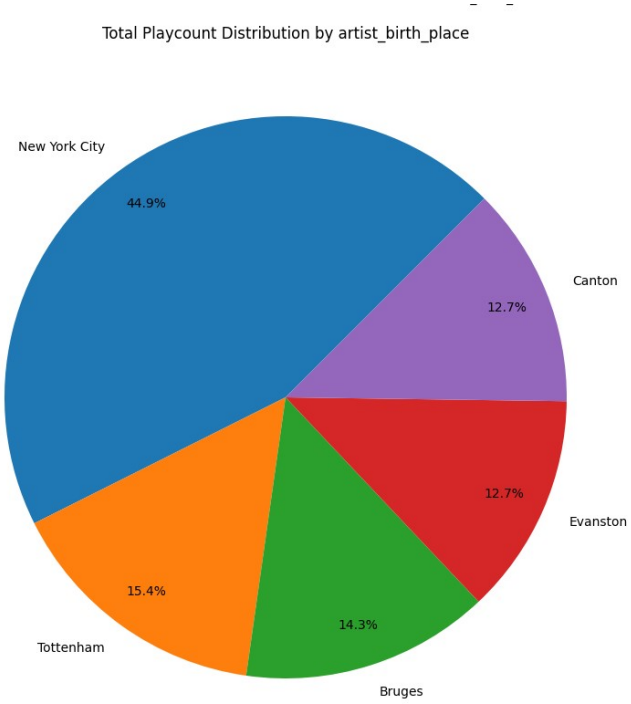

Total Playcount Distribution by artist_genre
Percentage of Total Playcount
0
10
20
30
40
50
60
Male
Female
artist_genre

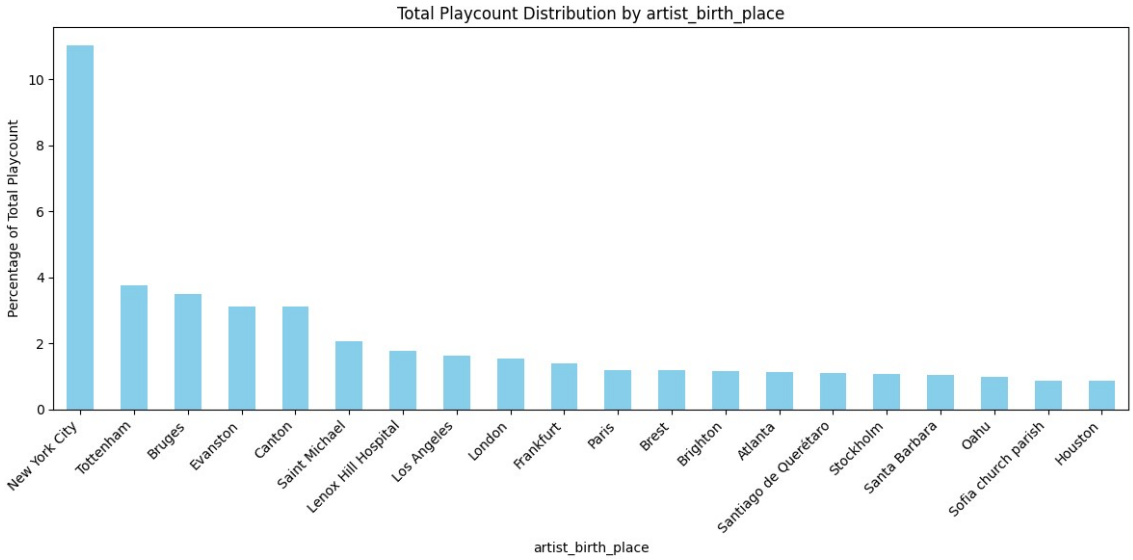
10M - Data di nascita:





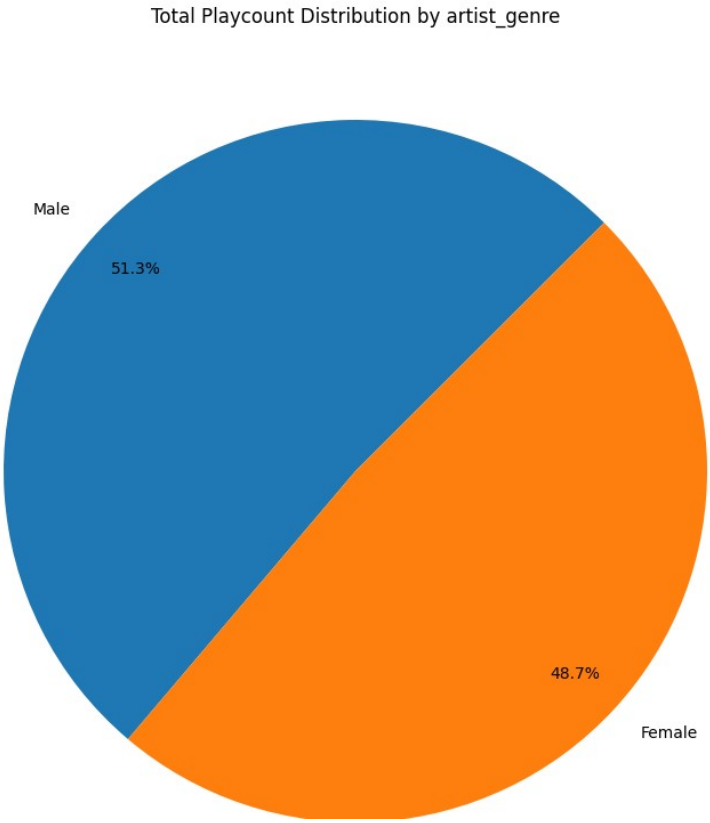
10M - Luogo di nascita:

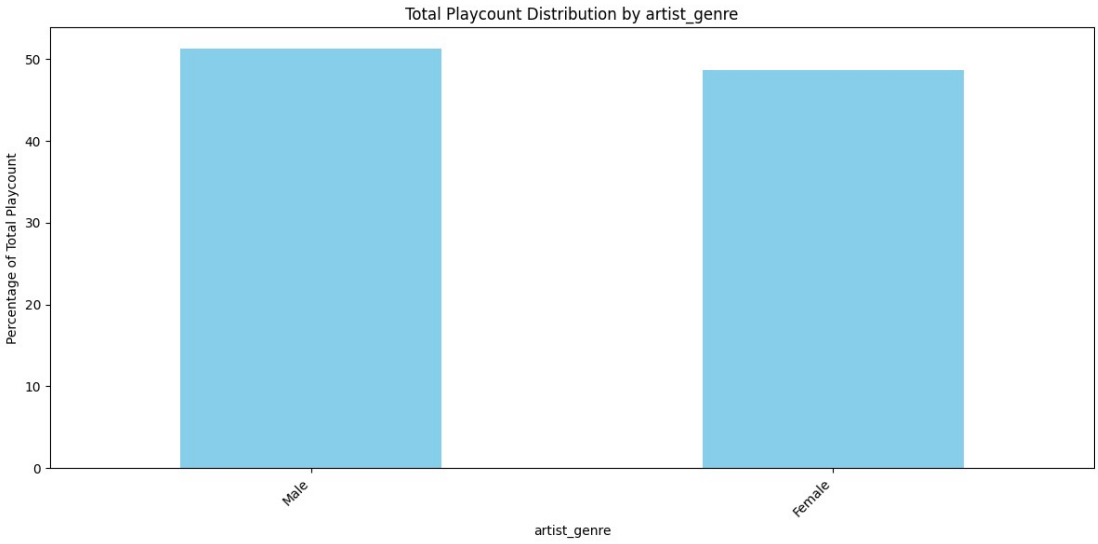




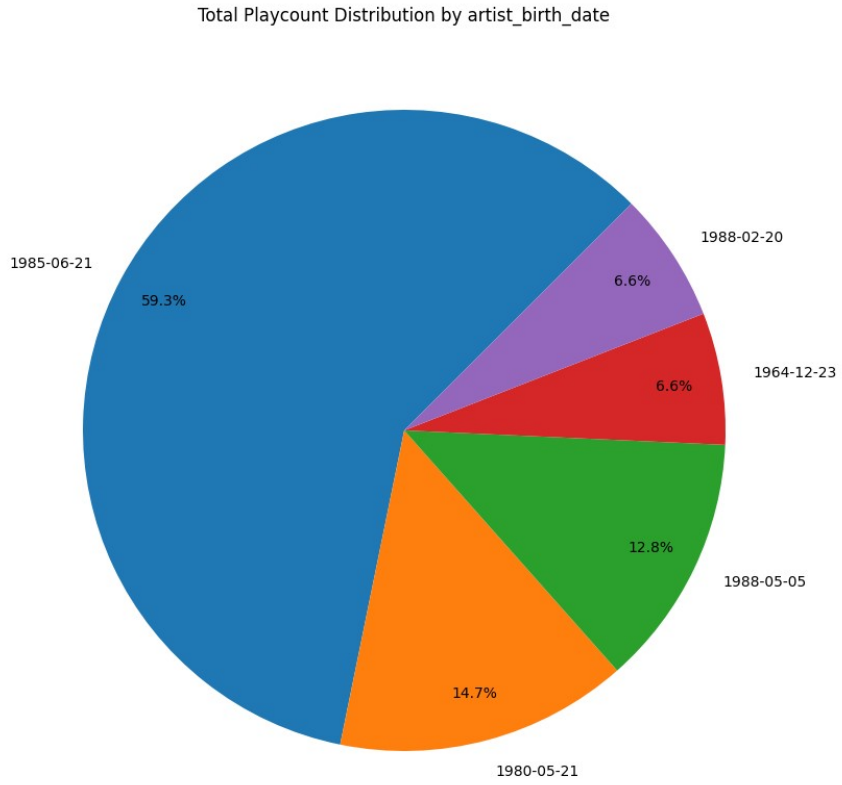
Per un campione più esaustivo di 10M, gli artisti ascoltati di più sono quelli nati a New York City negli anni ’80. Pur essendo i maschi quelli più ascoltati (59.4%), anche le femmine sono numerose (40.6%).

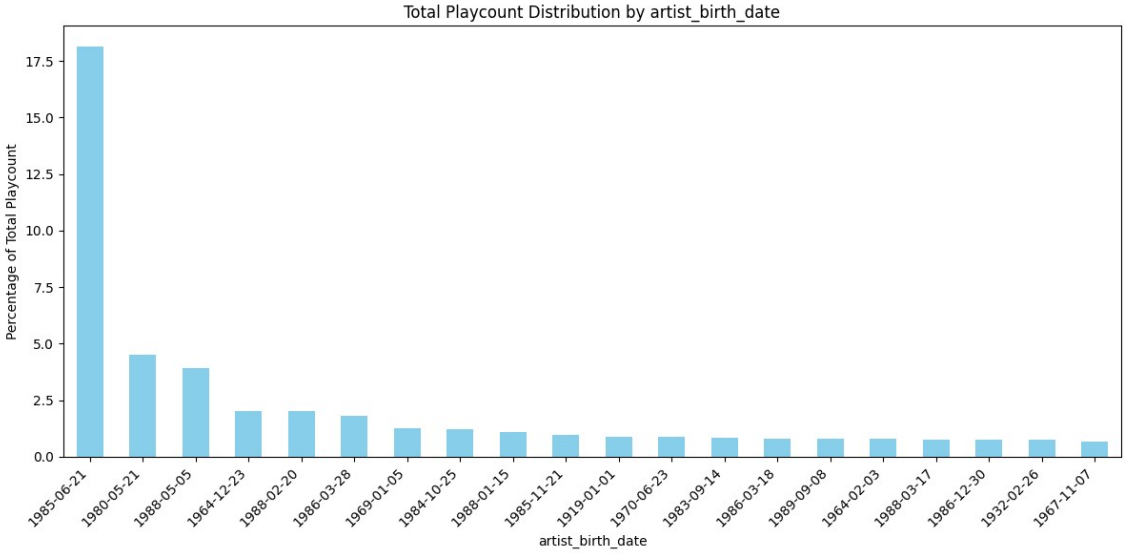
20M - Genere:


Total Playcount Distribution by artist_genre
Male
51.3%
48.7%
Female

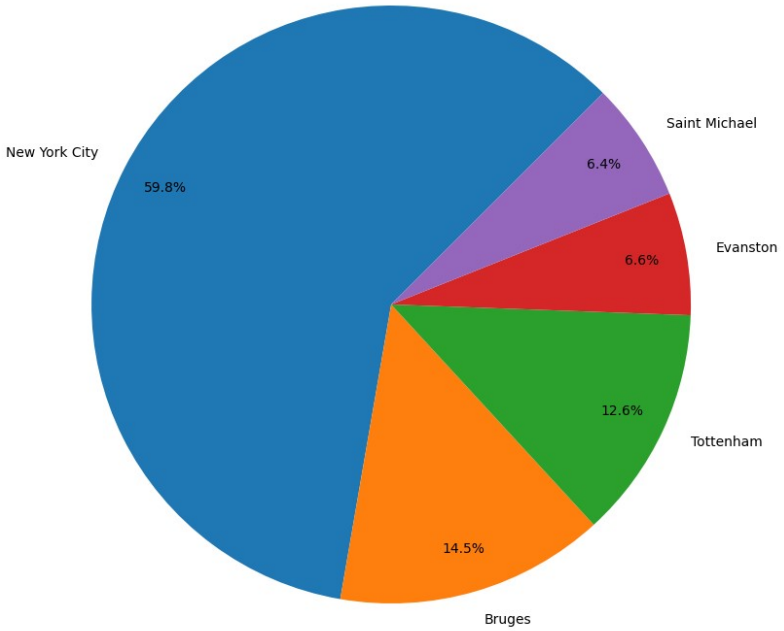

Total Playcount Distribution by artist_genre
Percentage of Total Playcount
0
10
20
30
40
50
Male
Female
artist_genre

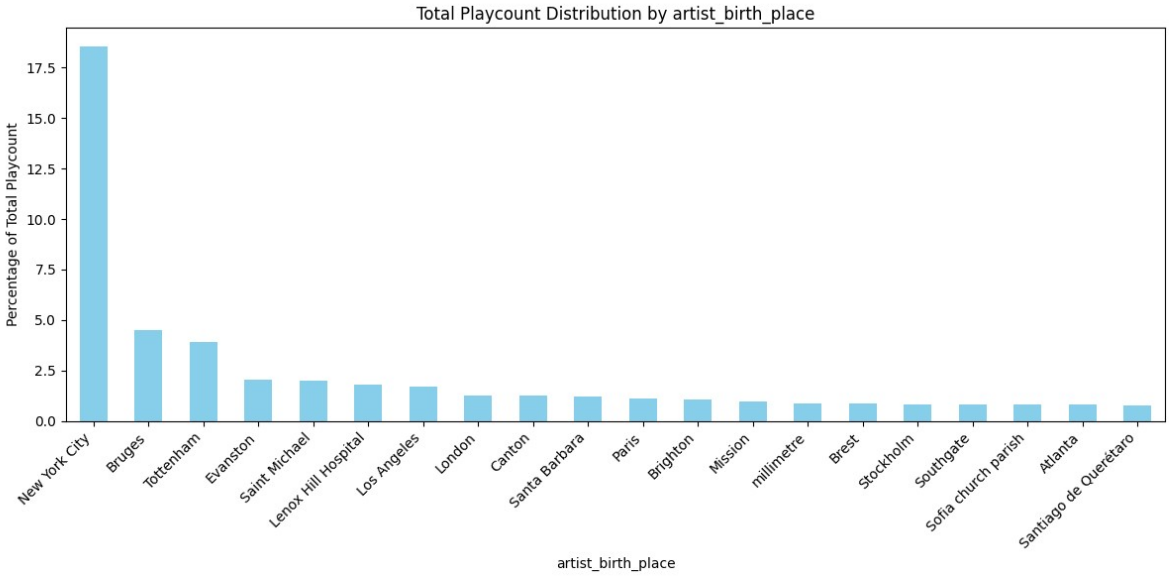
20M - Data di nascita:





20M - Luogo di nascita:





Si riconferma la maggioranza di artisti nati a New York City per un buon 60%, nati negli anni ’80, seguiti da quelli nati negli anni ’60, sia maschi che femmine con prevalenza di maschi.

Capitolo 6: Conclusioni e prospettive future

6.1: Conclusione

Questa tendenza può avere diverse interpretazioni. Potrebbe riflettere un periodo specifico di crescita e sviluppo della scena musicale a New York City negli anni '80, influenzata da vari generi musicali e movimenti culturali. La prevalenza di artisti maschili potrebbe essere correlata a dinamiche più ampie nell'industria musicale, come la rappresentazione e le opportunità differenziate tra generi.

Questa analisi offre non solo una panoramica sulle preferenze degli utenti, ma sottolinea anche l'eterogeneità delle scelte musicali all'interno del campione. L'utilizzo di Apache Spark ha dimostrato di essere cruciale per gestire efficientemente un dataset di questa portata, consentendo di risparmiare significativamente tempo di elaborazione e fornendo risultati accurati. L'approccio graduale nell'aumento delle dimensioni del campione ha permesso di valutare l'efficienza del processo di arricchimento dati su diverse scale.

Questo studio non solo arricchisce il dataset originale con informazioni demografiche, ma fornisce anche uno spaccato interessante sulle dinamiche di ascolto degli utenti. Tali conoscenze possono essere utilizzate per personalizzare ulteriormente le raccomandazioni musicali, migliorando l'esperienza complessiva degli utenti nell'ambito della piattaforma.

6.2: Sviluppi futuri

Per affrontare sfide future e migliorare ulteriormente il progetto, diverse direzioni possono essere esplorate. In primo luogo, l'espansione delle variabili demografiche considerate, come la nazionalità o l'età degli utenti, potrebbe fornire una visione più completa delle preferenze musicali. L'integrazione di modelli di machine learning potrebbe migliorare la precisione delle previsioni, personalizzando ulteriormente le raccomandazioni.

Oltre a ciò, l'implementazione di feedback diretto dagli utenti, l'integrazione con piattaforme di streaming musicale e l'esplorazione dettagliata di sottogruppi specifici di utenti potrebbero arricchire ulteriormente l'esperienza dell'utente. Ottimizzare le prestazioni, garantendo sicurezza e privacy e collaborando con esperti musicali potrebbero essere elementi chiave per il successo continuato del progetto.

Inoltre, la creazione di visualizzazioni interattive e l'analisi delle tendenze temporali potrebbero rendere l'esplorazione dei dati più accessibile e informativa. Guardando al futuro, la continua evoluzione di questo progetto offre un ampio spazio per l'innovazione e il perfezionamento delle strategie di raccomandazione musicale basate sui dati.