Report on Big Data project

Tra Pixel e Dati: Un'Analisi delle Metriche dei Videogiochi

David Cohen - Mat. 0001085730
https://github.com/davidcohenunibo/BigDataProject

27 ottobre 2023

Indice

1 Introduzione

In questa relazione, ci concentreremo sull'analisi di un ampio dataset relativo a giocatori, giochi e applicazioni su Steam. Una descrizione dettagliata del dataset sarà fornita, seguita da una discussione su studi e risultati precedenti nel campo.

1.1 Descrizione del Dataset

Ho scelto un interessante dataset basato su dati riguardanti la piattaforma `Steam`. Acquisito attraverso le API di `Steam` e archiviato dalla `Brigham Young University`, accessibile attraverso la loro piattaforma web[1]. Questo set di dati è notevolmente complesso, comprendendo una vasta gamma di metriche e variabili quali: il numero di utenti attivi, il tempo di gioco cumulativo, dettagli sugli acquisti e dei prodotti e molti altri attributi interessanti. Tali dati hanno permesso un'analisi comportamentale degli giocatori, permettendo un'indagine dettagliata dei pattern comportamentali emergenti. Alcune delle scoperte chiave sono:

1.2 Risultati dello Studio

Nel 2016, uno studio[1] ha rivelato diversi risultati interessanti e statisticamente significativi. Alcune delle principali scoperte sono:

- il numero di amicizie è basso rispetto ad altri social network, ma la *maggior parte del tempo di gioco è dedicato a giochi multiplayer*;
- determinati *giocatori si concentrano nel massimizzare le statistiche di tempo di gioco o i traguardi*, mentre altri *collezionano giochi che non giocano*;
- forti *correlazioni di amicizia* tra i giocatori che tendono a stringere amicizia *con coloro che sono simili in termini di popolarità, tempo di gioco, denaro speso e giochi posseduti*;
- Il tempo di gioco varia significativamente nel corso della settimana per i singoli giocatori; il loro *tempo di gioco non è costante di giorno in giorno.*
- è stata rilevata una *correlazione moderata tra quanto un gioco è giocato e quanti traguardi offre.*

[1] `https://steam.internet.byu.edu/`

1.3 Motivazioni ed Obiettivi

La scelta di lavorare con questo dataset è stata dettata dal mio interesse nel dominio dei videogiochi e dalla curiosità di applicare direttamente le competenze acquisite nel corso attraverso dati che posso comprendere agilmente. Questo ha permesso di immergermi maggiormente sia nel progetto che nel ruolo di data analyst.

L'obiettivo è replicare alcune delle analisi effettuate nello studio[1] menzionato, ma su una scala più ridotta. Diversi angoli di indagine nuovi saranno anche esplorati.

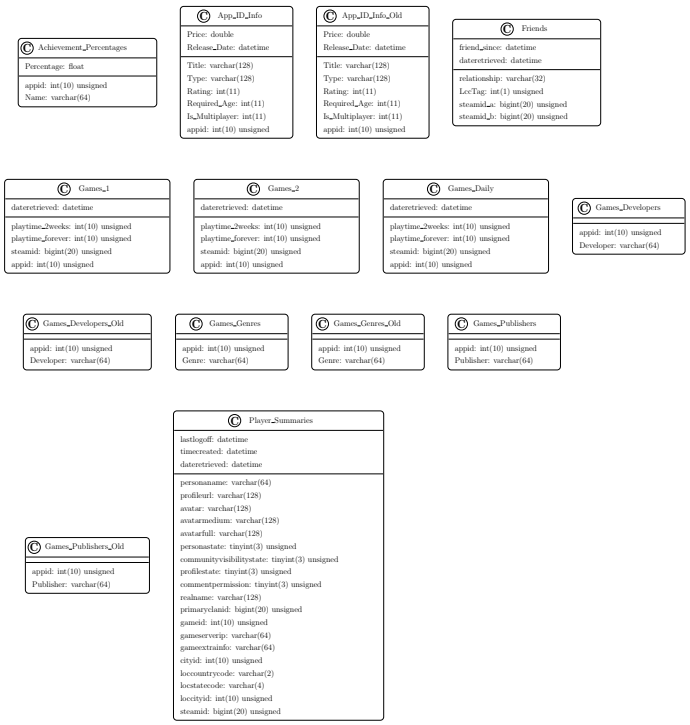
- **Bilancio tra socialità e gameplay**.
- **Tipologie di giocatori e le loro motivazioni**.
- **Variabilità del tempo di gioco durante la settimana**.
- **Correlazione tra traguardi offerti da un gioco e il suo tempo di gioco**.

2 Preparazione del dataset

Il dataset originale è un dump MariaDB di `160GB`. L'importazione del dataset è stata effettuata all'interno di un container Docker con un'istanza di MariaDB.

2.1 Esame e raffinamento

Dopo aver completato l'importazione, ho effettuato uno studio approfondito dello schema grezzo (Schema 1) del database. Il dump non presentava nessuna dipendenze tra le entity, seppur menzionate nel paper di riferimento. Durante questa fase, mi sono concetrato sull'identificazione della struttura e sull'associazione delle dipendenze, permettendomi di delimitare il mio dominio di analisi.

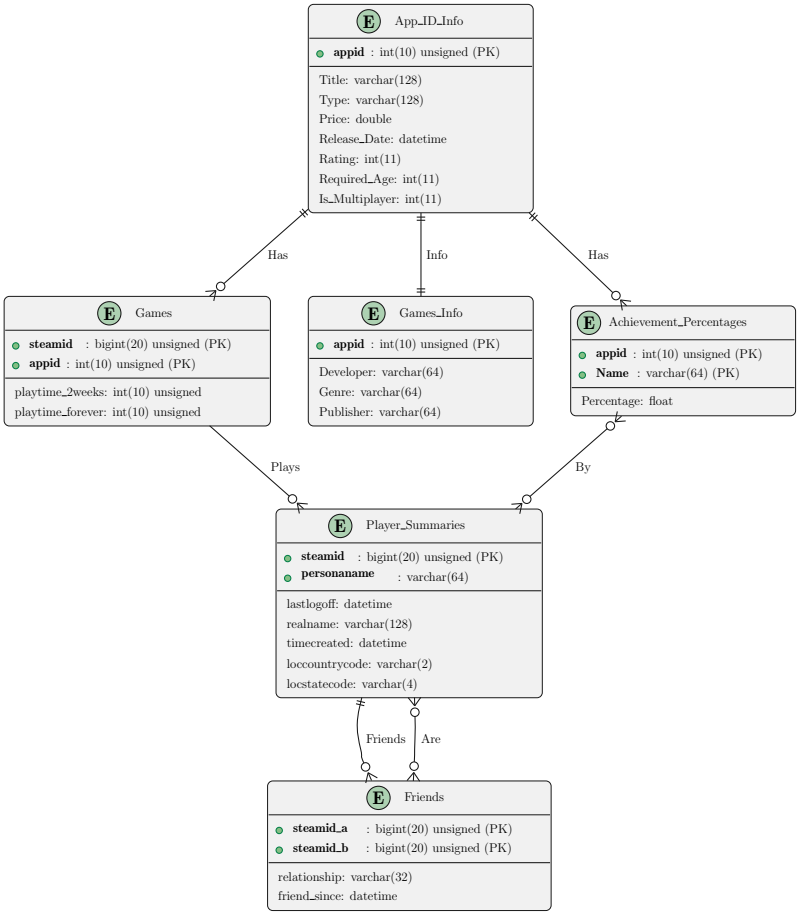


Schema 1: Schema raw del dump SQL importato.

Infine, ho deciso di ridurre notevolmente il dominio per poter fare una analisi più mirata e leggera sul dataset (Schema 2). Le modifiche principali includono:

- generi, sviluppatori ed editori sono ora in `Games_Info`;
- eliminate tabelle storicizzate come `App_Id_Info_Old`;

- `Games_Daily`.
- fusione di `Games_1` e `Games_2`;
- semplificazione delle chiavi composte;
- attributi centralizzati in `Games_Info`;
- modello denormalizzato per efficenza.



Schema 2: Schema entity-relationship rappresentante il dataset utilizzato. I cerchi piccoli rappresentano le chiavi primarie, mentre il cerchio verde grande rappresenta l'entità.

Ho effettuato uno snapshot delle varie fasi di raffinazione visionabili direttamente nella repository[2].

[2]https://github.com/davidcohenunibo/BigDataProject/tree/master/report/uml

2.2 Selezione e Riduzione dei Dati

Le fasi di riduzione e campionamento delle tabelle sono state automatizzate attraverso uno script `MySQL`. Tutte le query di selezione e riduzione sono state eseguite su tabelle temporanee per essere successivamente esportate in formato `CSV` per le analisi successive.Una dettagliata analisi dimensionale di ogni tabella è stata condotta al fine di selezionare e ridurre solo i dati pertinenti. Ad esempio, la tabella `Player_Summaries` è stata ridotta all `10%` (vedi Codice 1).

```
SET @max_id_steamid = (SELECT MAX(steamid) FROM Player_Summaries);
SET @min_id_steamid = (SELECT MIN(steamid) FROM Player_Summaries);
SET @range_steamid = @max_id_steamid - @min_id_steamid;
SET @limit_steam_id = ROUND(0.1 * @range_steamid);
SELECT @limit_steam_id AS limit_value;

CREATE TEMPORARY TABLE temp_Player_Summaries AS
SELECT * FROM Player_Summaries
WHERE steamid BETWEEN @min_id_steamid AND @min_id_steamid + @limit_steam_id;
```

Codice 1: Sampling del `10%` sul numero totale di players.

Allo stesso modo, altre tabelle di considerevole dimensione, come le precedenti versioni di `Games_1` e `Games_2`, sono state ridotte e poi combinate in una unica entità principale `Games` (Codice 2).

```
CREATE TEMPORARY TABLE temp_Games AS
SELECT g.*
FROM (
    SELECT g1.*
    FROM Games_1 g1
    WHERE g1.appid IN (SELECT appid FROM temp_App_ID_Info)
    AND g1.steamid IN (SELECT steamid FROM temp_Player_Summaries)

    UNION

    SELECT g2.*
    FROM Games_2 g2
    WHERE g2.appid IN (SELECT appid FROM temp_App_ID_Info)
    AND g2.steamid IN (SELECT steamid FROM temp_Player_Summaries)
) g;

CREATE TEMPORARY TABLE temp_Games_Info AS
SELECT gd.*, gp.Publisher, gg.Genre
FROM Games_Developers gd
JOIN temp_App_ID_Info ai ON gd.appid = ai.appid
JOIN temp_Games_Publishers gp ON gd.appid = gp.appid
JOIN temp_Games_Genres gg ON gd.appid = gg.appid;
```

Codice 2: `Union` delle sessioni di gioco inserite in una `TEMPORARY TABLE`.

2.3 Preparazione dei dati

I file sono disponibili nei buckets `AWS S3`. I percorsi utilizzati sono gli stessi presente nel progetto `project.ipynb` situato nella `home` del workspace `bigdataworkspace`.

```
val bucketname = "unibo-bd2324-dcohen"

val path_apps = "s3a://"+bucketname+"/datasets/project/App_ID_Info.csv"
val path_players = "s3a://"+bucketname+"/datasets/project/Player_Summaries.csv"
val path_friends = "s3a://"+bucketname+"/datasets/project/Friends.csv"
val path_games_full = "s3a://"+bucketname+"/datasets/project/Games.csv"
val path_games = "s3a://"+bucketname+"/datasets/project/Games20Percent.csv"
val path_games_daily = "s3a://"+bucketname+"/datasets/project/Games_Daily.csv"
val path_games_info = "s3a://"+bucketname+"/datasets/project/Games_Info.csv"
val path_achievement =
↪  "s3a://"+bucketname+"/datasets/project/Achievement_Percentages.csv"
```

2.4 Pre-elaborazione dei dati

Per assicurarmi che i dati siano adatti all'analisi, è stato necessario un processo di pre-elaborazione.

- **Estrazione dei dati:** i dati sono suddivisi in campi riga per riga usando espressioni regolari, specificatamente `commaRegex` e `pipeRegex`, per gestire vari delimitatori come virgole e simboli di `pipe`.
- **Pulizia dei campi:** i campi estratti vengono puliti da eventuali virgolette e spazi bianchi in eccesso usando il metodo `cleanField`.
- **Conversione sicura:** il processo di conversione dei campi testuali in tipi di dati appropriati (come `Long`, `Double`, `Int`, ecc.) viene gestito in modo sicuro attraverso il metodo `safelyConvert`.
- **Conversione delle date:** le date in formato stringa vengono convertite nel formato `"yyyy-MM-dd HH:mm:ss"` e poi estratto solo l'anno usando il metodo `yearFromTimestamp`, come visto a lezione. Il formato della data è stato specificato mediante la classe `SimpleDateFormat`.
- **Parsing dei dati:** diversi metodi di parsing sono stati implementati per ogni tipo di dato. Ogni metodo di parsing restituisce un'opzione, che potrebbe essere `None` se la riga non può essere analizzata correttamente. In tal caso, la riga viene scartata.
- **Gestione dei valori mancanti:** nel caso di `parsePlayerSummaryLine`, vengono gestiti i valori mancanti per `realname`, `loccountrycode` e

`locstatecode` fornendo un valore predefinito (`"Unknown"`) se non sono presenti.

3 Query

Dopo aver svolto **14 query** diverse, mirate ad esplorare gli aspetti analizzati nella ricerca [1] e alcune mie aggiunte personali, ho scelto di focalizzarmi su **tre query specifiche** per la relazione. Le query selezionate sono:

1. **Correlazione tra il numero di amici e il numero di applicazioni multiplayer**.
2. **Tempo medio di gioco in rapporto all'anno di rilascio**.
3. **Applicazioni con la percentuale di completamento più elevata**.

L'insieme completo delle query è disponibile nella repository[3].

4 Implementazione delle Query

Prima di procedere con la presentazione delle specifiche delle singole query, vorrei riassumere alcune ottimizzazioni e design patterns comuni che ho adottato durante lo sviluppo del progetto:

- **Campionamento dei Dati:** per gestire l'overhead, ho applicato una strategia iniziale esclusivamente sulle sessioni di gioco dato il numero elevato di elementi. Per farlo, ho utilizzato il metodo `sample(false, 0.3)` per campionando il 30% dei dati e accelerando significativamente la fase di testing.

```
val gameSessions =
↪  sc.textFile(path_games).flatMap(SteamGamesParser.parseGameLine).sample(false, 0.3)
```

- **Broadcasting di App:** dopo aver valutato il volume dei dati con `count()` su tutti gli `RDD`. Sulla base dei conteggi e della frequenza di utilizzo, ho selezionato l'`RDD` `apps` per il broadcasting. Questo sfrutta le capacità di `broadcast()` per minimizzare lo shuffling dei dati e ridurre l'overhead.

[3] `https://github.com/davidcohenunibo/BigDataProject/tree/master/spark/src`

```scala
val appsMap = apps.mapValues(/*...*/).collectAsMap().toMap
val broadcastedApps = sc.broadcast(appsMap)
```

4.1 Ottimizzazioni Generali delle Prestazioni

- **Monitoraggio con Spark UI**: durante la sperimentazione ho utilizzato `Spark UI` per un'analisi dettagliata delle fasi dei singoli job, che mi hanno permesso di confermare le modifiche di prestazioni adottate per ogni singola query.
- **Minimizzazione dei Mapping**: ho pensato di dividere l'`RDD` `gameSessions` con chiave composta in due nuovi `RDD` `gameSessionsWithAppKey` e `gameSessionsWithSteamKey`, facilitando le query mirate e ottimizzando le prestazioni.
- **Ottimizzazione del Partizionamento dei Dati**: per migliorare l'efficienza nelle operazioni di `join()`, ho usato un HashPartitioner con `core*2` partizioni attraverso `partitionBy()`. In generale, tutti gli `RDD` iniziali sono stati partizionati prima del caching.
- **Coalescenza Pre-Output**: utilizzo di `coalesce(1)` per consolidare le partizioni prima dell'output, migliorando l'efficienza nella fase di stampa o di salvataggio dei dati.
- **Ottimizzazione del Traffico di Rete e Caching**: attivazione del caching per gli `RDD` frequentemente utilizzati oltre a quelli iniziali quale `playerCounts` e `totalPlaytimeForGame`.

4.2 # Correlazione tra Tempo Totale di Gioco e Numero di Amici

Obiettivo della Query: esaminare la correlazione tra la dimensione della rete sociale di un giocatore e la quantità di giochi multiplayer che possiede.

4.2.1 Dettagli Implementativi

Dettaglio	Valore
File di Input	`/project/{Games.csv, Friends.csv}`
File di Output	`/project/correlation/`
Tempo di Esecuzione	`9,11 secondi`

4.2.2 Strategia Adottata

Gli `RDD` fondamentali in questa analisi sono:

- `gameSessionsWithAppKey`: informazioni sulle sessioni di gioco associate a ciascuna applicazione.
- `multiplayerSessions`: sessioni di gioco multiplayer.
- `playerCounts`: numero di amici associati a ciascun giocatore.
- `broadcastedApps`: dati rilevanti da `apps`, distribuiti a tutti i nodi di calcolo.

Fasi di Sviluppo:

1. **Filtraggio e Aggregazione**: filtro le sessioni in `gameSessions` per selezionare quelle multiplayer, per poi aggregarle per utente.
2. **Correlazione**: eseguo un join tra `multiplayerSessions` e `playerCounts`, e calcolo la correlazione tra il numero di amici e le sessioni multiplayer.

Considerazioni sulle Prestazioni:

- **Ottimizzazione dell'Ordine delle Operazioni**: le trasformazioni sono state riorganizzate per posizionare `filter()` prima di `reduceByKey()`, riducendo così la quantità di dati da processare e ottimizzando i tempi.

4.2.3 Visualizzazione e Discussione dei Risultati

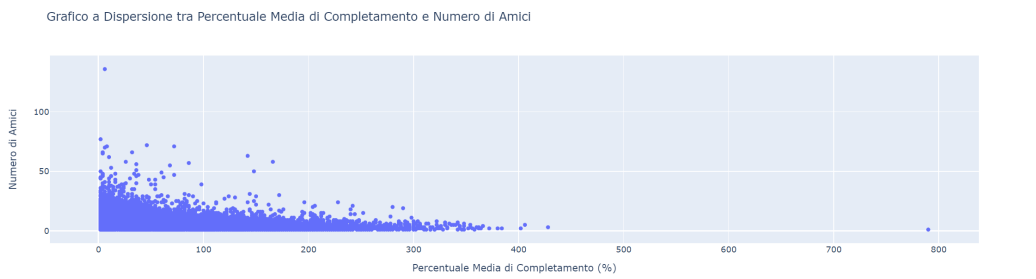


Figura 3: Grafico a Dispersione tra Percentuale Media di Completamento e Numero di Amici.

La correlazione tra `average_percentage` e `friendcount` è di **0.0748**, indicando una correlazione positiva molto debole.

Non esiste una relazione lineare forte tra le due variabili basandosi su questo campionamento. Tuttavia, dato che i dati rappresentano solo il 10% del totale, ci potrebbero essere alcune alterazioni e sarebbe opportuno estendere l'analisi all'intero dataset per conclusioni più accurate.

4.3 ## Giochi con più obiettivi completati

Obiettivo della Query: identificare in quale gioco i giocatori cercano di completare più obiettivi.

4.3.1 Implementazione in Spark

Dettaglio	Valore
File di input	`/project/ { Achievement_Percentages.csv, App_ID_Info.csv }`
File di output	`/project/most_achieved_app/`
Tempo di esecuzione	`3 secondi`

4.3.2 Strategia Adottata

Gli `RDD` chiave nel processo sono le seguenti:

- `achievementsWithAppKey`: collezione delle percentuali di completamento degli obiettivi per ogni applicazione.
- `totalAndCountAchievement`: somma e conteggio delle percentuali di completamento per ciascuna applicazione.
- `averageAchievement`: media delle percentuali di completamento per ciascuna applicazione.
- `mostAchievedAppsAverage`: applicazioni ordinate in base alla media delle percentuali di completamento.
- `broadcastedApps`: dati rilevanti da `apps`, distribuiti a tutti i nodi di calcolo.

Fasi di Sviluppo:

1. **Calcolo Somma e Conteggio delle Percentuali**: utilizzo `mapValues()` e `reduceByKey()` per calcolare la somma e il conteggio delle percentuali di completamento per ogni applicazione in `achievementsWithAppKey`.
2. **Calcolo Media delle Percentuali**: applico una nuova trasformazione `mapValues()` a `totalAndCountAchievement` per calcolare la media delle percentuali di completamento.

3. **Ordinamento e Dettagli dell'App**: utilizzo `map()` per unire le medie delle percentuali con i dettagli delle applicazioni da `broadcastedApps`, e poi ordino il risultato con `sortByKey()`.

Considerazioni sulle Prestazioni:

- **Ottimizzazione dell'Aggregazione**: utilizzo `reduceByKey()` per aggregare efficacemente le percentuali, riducendo la quantità di dati da trasferire.

4.3.3 Visualizzazione e Discussione dei Risultati

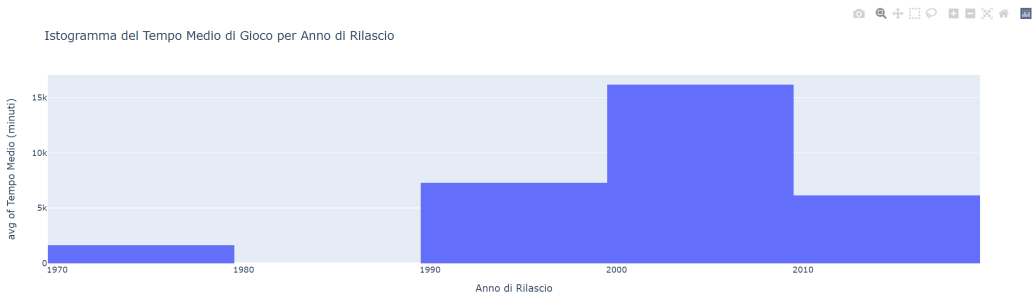


Figura 4: Istogramma del Tempo Medio di Gioco per Anno di Rilascio.

Il tempo medio di gioco è aumentato significativamente dagli anni '70 agli anni 2010, con un **picco nei primi anni 2010**. La diminuzione verso la fine del decennio potrebbe essere dovuta all'avvento dei giochi per dispositivi mobili e alla saturazione del mercato.

4.4 ### Media del tempo di gioco in base alla data di rilascio

Obiettivo della Query: In questa query, l'obiettivo è di calcolare il tempo medio di gioco in base all'anno di rilascio del gioco.

4.4.1 Implementazione in Spark

Dettaglio	Valore
File di input	`/project/ { Games.csv, App_ID_Info.csv }`
File di output	`/project/avg_playtime_release_year/`
Tempo di esecuzione	`1 secondo`

4.4.2 Strategia Adottata

Le variabili chiave nel processo sono le seguenti:

- `gameSessionsWithAppKey`: informazioni sulle sessioni di gioco associate a ciascuna applicazione.
- `avgPlaytimeByReleaseYear`: tempo medio di gioco per ogni anno di rilascio.
- `broadcastedApps`: dati rilevanti da `apps`, distribuiti a tutti i nodi di calcolo.

Fasi di Sviluppo:

1. **Mappatura dell'Anno di Rilascio e del Tempo di Gioco**: utilizzo `map()` per associare l'anno di rilascio di ogni gioco al relativo tempo di gioco utilizzando i dati rilevanti in `broadcastedApps`.
2. **Aggregazione del Tempo di Gioco**: applico `aggregateByKey()` per calcolare la somma e il conteggio del tempo di gioco per ogni anno di rilascio.
3. **Calcolo del Tempo Medio di Gioco**: utilizzo `mapValues()` per calcolare il tempo medio di gioco, dividendo la somma totale per il numero di conteggi.

Considerazioni sulle Prestazioni:

- **Ottimizzazione dell'Aggregazione**: `aggregateByKey()` è stato utilizzato per un'aggregazione efficiente, minimizzando i trasferimenti di dati su rete.
- **Broadcasting efficente**: ho riutilizzato la variabile `broadcastedApps` per evitare il caricamento ridondante dei dettagli riguardo alle applicazioni e ho effettuato direttamente il mapping desiderato con `releaseYear` come chiave primaria.

4.4.3 Visualizzazione e Discussione dei Risultati

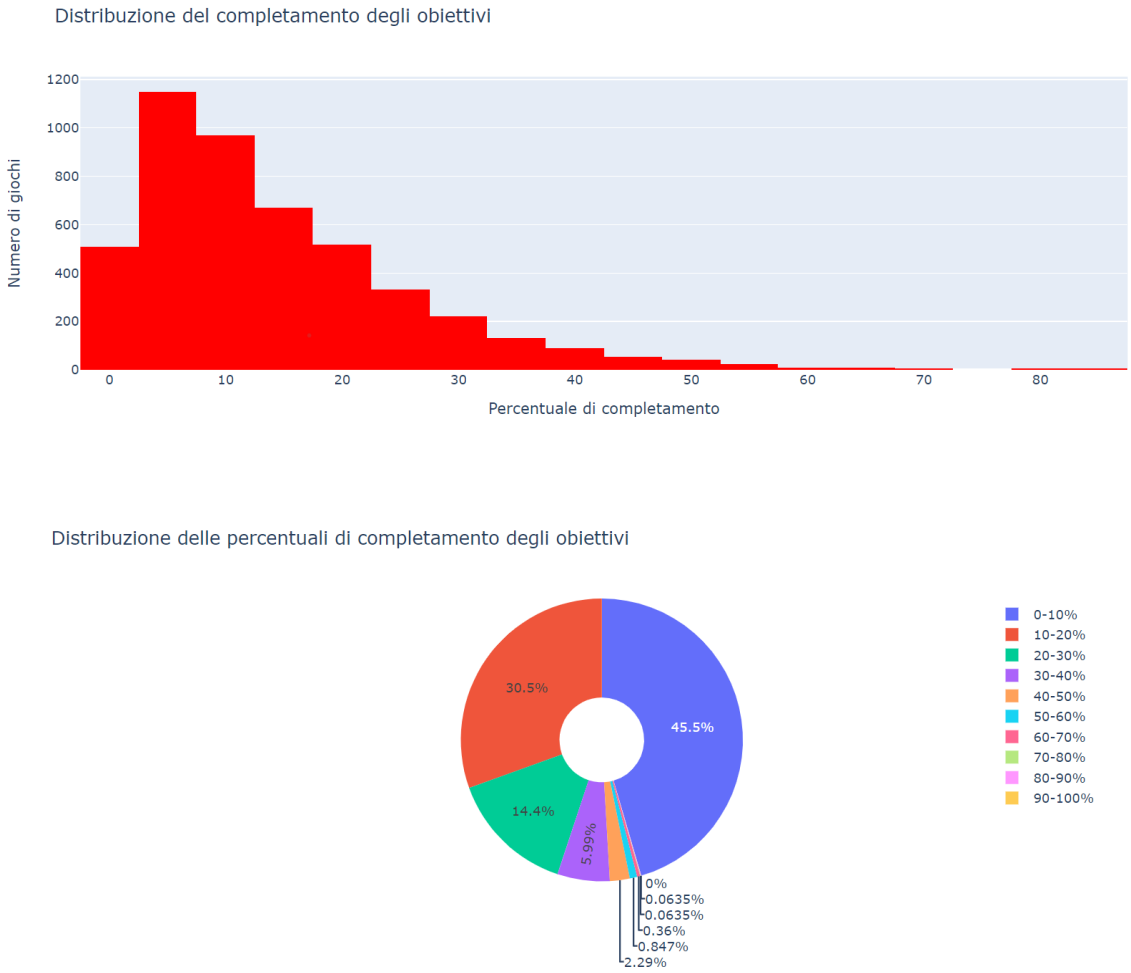


Figura 5: Distribuzioni delle percentuali di completamento degli achievements.

Dall'istogramma e dal grafico a torta, emerge che:

- Il **45.5%** dei giochi ha una percentuale di completamento tra **0-10%**.
- Il **30.5%** dei giochi ha un completamento tra **10-20%**.
- Il **14.4%** rientra nella fascia del **20-30%**.
- Percentuali più basse, come **5.9%**, indicano completamenti tra **30-40%** e così via.

- Giochi con completamento oltre il **70%** sono molto rari, come indicato dalle piccole fette nella parte inferiore del grafico.

Queste statistiche confermano che molti giochi non vengono completati in grande misura dai giocatori e **solo una piccola percentuale di giochi raggiunge alti livelli di completamento**.

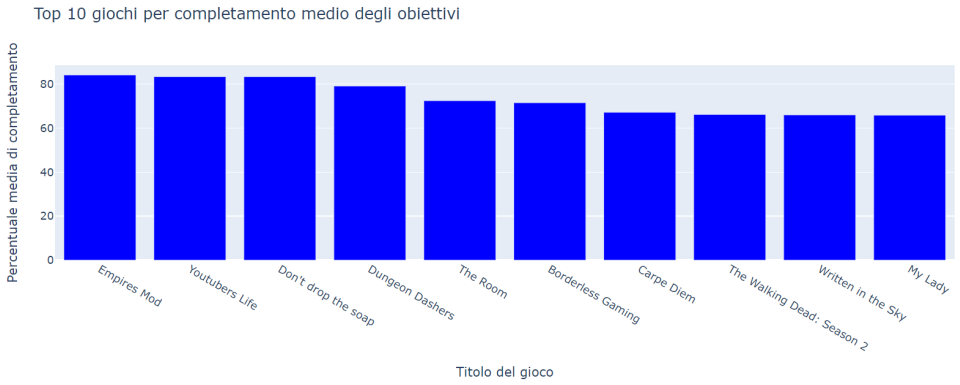


Figura 6: Istogramma con i top 10 giochi con più achievements.

Riferimenti bibliografici

[1] Mark O'Neill, Elham Vaziripour, Justin Wu, and Daniel Zappala. Condensing steam: Distilling the diversity of gamer behavior. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, page 81–95, New York, NY, USA, 2016. Association for Computing Machinery.