A DATA WAREHOUSE IMPLEMENTATION FOR TENNIS MATCHES ANALYSIS

STUDENTS

Samuele Cucchi, Francesco Gemignani

LABORATORY OF DATA SCIENCE PROJECT

Data Science & Business Informatics

Academic Year 2020/2021

Sommario

Il dataset che ci è stato affidato contiene le informazioni che riguardano l'analisi di differenti matches di tennis dei principali tornei che intercorrono dal 2016 al 2021. Il dataset è composto da 186.073 record e 49 features. Gli attributi riguardano le performaces dei giocatori, le statistiche della partita, il torneo a cui partecipano, la data e le informazioni sulla provenienza dei giocatori.

Nella **prima parte** del progetto abbiamo il compito di creare e popolare un data warehouse nel rispetto dello schema che ci è stato fornito, il cui singolo evento descrive un determinato match, giocato tra due giocatori. Nella **seconda parte** abbiamo definito un data flow con SSIS per rispondere a determinate business questions, mentre nell'**ultima sezione** abbiamo utilizzato SQL Analysis Services per costruire un OLAP cube e successivamente sviluppato queries MDX per interrogarlo. Infine sono state sviluppate delle dashboard con Power BI per la visualizzazione dei dati.

Il progetto è stato sviluppato in Python 3.7, utilizzando Pycharm come ambiente di sviluppo e Anaconda per l'analisi e il preprocessing dei dati. L'intero codice è stato sviluppato senza il supporto della libreria pandas, utilizzata soltanto per la fase di analisi e preprocessing dei dati. Come DBMS ci siamo serviti di Microsoft SQL Server Management Studio. Le ultime due parti sono state realizzate con le estensioni SQL Server Integration Services (SSIS) e SQL Server Analysis Services (SSAS) di Visual Studio 2019.

Parte 1

I dati con cui creare e popolare il database ci sono stati forniti in 4 differenti file csv: *tennis.csv*, *male_players.csv*, *female_players.csv* e *geography.csv*. Inoltre, abbiamo utilizzato ulteriori dati per reperire le informazioni dei continenti e linguaggi mancanti.

Assignment 0

Abbiamo creato lo schema utilizzando Microsoft SQL Server Management Studio sul server lds.di.unipi.it. Lo snowflake schema è composto dalla fact table e da 4 dimension tables. Ogni tabella contiene una propria chiave primaria non nulla. Al fine di garantire l'integrità tra i valori, anche le chiavi esterne non contengono valori nulli. Come possiamo notare, non trattandosi di uno star schema, sono presenti chiavi esterne anche nelle dimension tables (*Tournament* e *Player*).

Considerando che gli attributi non numerici sono stringhe con una lunghezza molto variabile (eg. *match_id* e *round* possono avere più di 25 caratteri di differenza), abbiamo deciso di rappresentarli con *nvarchar*, mentre i restanti valori con il formato numerico appropriato (*int* e *real*).

Assignment 1

In questo task ci è stato richiesto di ripartire il contenuto del file *tennis.csv* nelle rispettive tabelle. Per realizzare ciò, abbiamo generato un file csv per ogni tabella dello schema, essi sono: *match.csv*, *tournament.csv*, *date.csv*, *player.csv*, e *geography.csv*. Ogni file è quindi rappresentativo della tabella a cui è associato. L'algoritmo di splitting, contenuto nel file *split2Schema.py*, genera le tabelle scansionando un'unica volta il file *tennis.csv*. Per ogni record letto, la procedura genera online le righe di ogni tabella, delegandone la costruzione alla procedura associata. L'algoritmo per ogni riga, effettua i seguenti passi:

1. Estrazione o Derivazione della(delle) chiave(i) associata(e) alla tabella.
   In particolare, per ogni record letto del file *tennis.csv* possiamo avere al più una chiave per le dimensioni *Match*, *Tournament* e *Date* mentre al più 2 chiavi per *Player* e *Geography*, di queste una per il vincitore ed una per il perdente, sia per quanto riguarda le informazioni del giocatore, sia per la sua provenienza.

   Nel caso delle tabelle *Match* e *Tournament*, le chiavi vengono derivate concatenando informazioni relative al torneo ed il livello.

2. Verifica della presenza della(e) chiave(i) nel keyset associato a quella tabella. Ovviamente, considerando l'unicità della chiave primaria, genero e inserisco quei record la cui chiave non è presente nel keyset. I keyset sono implementati utilizzando i *set*.

3. Costruzione del(i) record(s) per quella tabella. La costruzione e l'inserimento della riga avviene se la verifica precedente ha dato esito negativo. In particolare, per ogni tabella, posso avere da 0 a 2 inserimenti per ogni record letto. Ad ogni table è associato un dizionario, in modo tale da gestire sia il mapping degli attributi che il tipo. Gli attributi non presenti sono derivati e calcolati online (ie. *quarter*, *day*, *month*, *year*, *sex*, *year_of_birth* assoluto, etc..).

L'algoritmo prima di iniziare la scansione di tutti i record, crea i dizionari per derivare il sesso di ogni player e la nazionalità (continente e language) in base ai dati che ci sono stati forniti. Questi dizionari sono utilizzati online durante la fase di costruzione dei record. L'utilizzo di strutture dati primitive come i dizionari e i set, insieme al fatto di scansionare univocamente i dati ci ha permesso di garantire una buona efficienza. L'intera fase di splitting e la generazione dei 5 file csv impiega una media di $10,34$ sec.: risultato ottenuto su un MBP 2015 - 2,2 GHz Intel Core i7 quad-core - 16 GB 1600 MHz DDR3, virtualizzando Windows 10 con il sw Parallels. La dimensione dei file csv in questo caso non è stata d'impatto (ca. 24 MB). Nel caso in cui tale valore fosse stato penalizzante avremmo potuto valutare scelte implementative differenti, come ad esempio generare e inserire i record nelle tabelle senza passare dai csv. In tal caso avremmo sicuramente ottenuto maggiori benefici in termini di spazio ma, probabilmente, compromettendone la complessità in tempo di esecuzione.

Analisi & Preparazione dei Dati

In questa fase, abbiamo analizzato i files generati dopo il processo di splitting appena descritto. La maggior parte dei valori mancanti l'abbiamo riscontrata negli attributi che descrivono le statistiche della partita nel file *match_csv*, dove il circa il 52% dei dati non è presente. In generale abbiamo preferito non eliminare i record, ma piuttosto sostituirne i valori in modo tale non avere null values nel database.

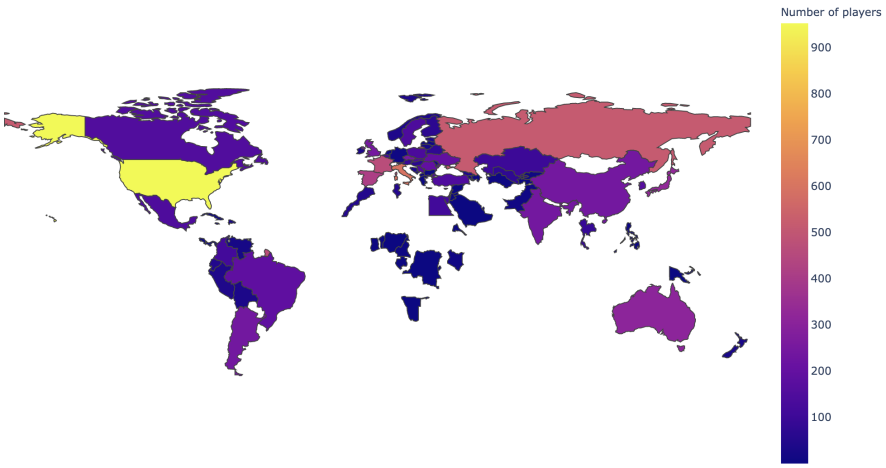

Figure 1: Mappa coropletica per mostrare le origini dei giocatori

I missing values di attributi discreti sono stati sostituiti considerando la moda, mentre per quelli numerici abbiamo scelto il valore mediano, il quale non tiene conto di eventuali valori anomali che sono presenti nei dati. Inoltre, abbiamo riscontrato che esistono dei giocatori che non sono presenti nei due files che contengono il sesso, *male_plater.csv* e *female_player.csv*. Il sesso è stato sostituito con il valore più frequente. Anche l'altezza dei giocatori è stata corretta considerando il valore mediano raggruppandolo in base al sesso. Il file *geography.csv* contiene

alcuni continenti e linguaggi mancanti. Questo avviene perchè sono stati trovati *country_id* non presenti nel file *country_list* che ci è stato fornito. In questo caso, abbiamo utilizzato due ulteriori file csv in modo tale da rimpiazzare tali valori con il continente e il linguaggio più appropriato. I paesi rimanenti con valori nulli sono stati sostituiti selettivamente, perchè le sigle contenevano errori di battitura che ne impedivano la correzione automatica (ie. ITS invece di ITA).

Dopo aver sostituito i valori mancanti abbiamo ricercato la presenza di anomalie sia nei valori numerici che nelle stringhe. Abbiamo rilevato diversi errori di battitura nell'attributo *score* (e.g. singoli caratteri non unicode, blank spaces non nulli) e nelle altezze dei giocatori (e.g. ht=2cm). Inoltre, abbiamo riscontrato che esistono partite che durano 0 minuti nonostante abbiamo uno score completo (match non annullato). Anche in questo caso, come per i missing values, abbiamo gestito le anomalie sostituendole con il valore mediano o quello più frequente.

Infine, abbiamo rilevato molti valori duplicati di *tourney_id*. Per ovviare a questo problema abbiamo deciso di concatenare a tale valore il *tourney_level*. Lo stesso è stato fatto anche sulla chiave primaria di *Match*, la quale concatena al *match_id* il nuovo *tourney_id*. Distinguere i tornei (e i match) anche in base al livello ci ha permesso di incrementare sensibilmente il numero di record sia della fact table che di *Tournament*.

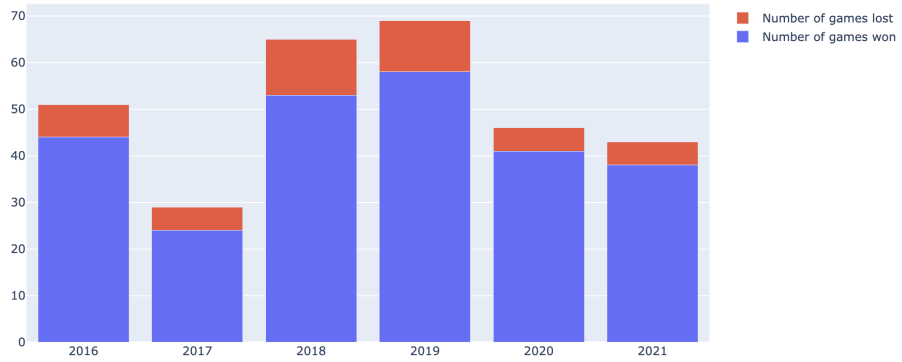

Figure 2: Andamento delle partite vinte/perse da Novak Djokovic negli anni

Una volta terminata la fase di preparazione, i record dei file csv sono pronti per essere caricati nelle rispettive tabelle.

Assignment 3

In questo task abbiamo il compito di popolare le tabelle create inizialmente su SQL Server. Dopo aver stabilito la connessione con il database del gruppo che ci è stato assegnato, sul server lds.di.unipi.it abbiamo caricato tutti i record per ogni tabella. Abbiamo deciso di caricare i dati in chunk per non sovraccaricare il server di queries. La dimensione dei blocchi varia in base al numero di record contenuti nel rispettivo file, in particolare si è scelta una dimensione del 5% rispetto al numero di righe. Abbiamo prima caricato le child dimension tables dello snowflake schema e poi le parent dimension così da non riscontrare errori con le relazioni tra chiavi esterne e primarie non presenti (le PK delle child dimension devono essere già presenti quando inserisco la FK della parent dimension).

Parte 2

In questa parte abbiamo utilizzato l'estensione SQL Server Integration Services (SSIS) di Visual Studio per rispondere ad alcune business question. Per ogni assigment sono stati sviluppati data flows che utilizzano soltanto nodi SSIS standard, senza il supporto di comandi SQL all'interno dei moduli.

Assignment 0

For every year, the tournaments ordered by spectators.

Abbiamo utilizzato il nodo *Origine OLE DB* per accedere alla tabella *Tournament*, filtrando soltanto gli attributi necessari per rispondere alla domanda. Essi sono: *tourney_id*, *date_id* e *tourney_spectators*. La join con la tabella Date, necessaria per avere il campo year, è stata fatta con il modulo *Ricerca*. A questo punto ogni record contiene tutte le informazioni necessarie per rispondere alla domanda. Prima di scrivere i dati sul file *ass0_results* tramite il nodo *Destinazione file flat*, abbiamo ordinato i dati per anno (crescentemente) e per numero di spettatori (decrescentemente), utilizzando il modulo *Ordinamento*.

Assignment 1

For any given player, his or her "nemesis" is the player against whom he or she lost most matches. List every player with the respective nemesis and the number of matches lost.

Abbiamo eseguito l'accesso alla tabella *Match* utilizzando il modulo *Origine OLE DB*, filtrando soltanto le colonne relative alle chiavi esterne: *winner_id* e *loser_id*. Per contare il numero di volte in cui un giocatore ha perso contro ogni suo avversario abbiamo utilizzato la funzione *Aggregazione*, raggruppando rispettivamente per *loser_id* e *winner_id*, poi contando il numero di record. Il risultato dell'aggregazione è contenuto nella colonna *num_defeats*. Successivamente, per reperire i nomi dei giocatori del match, abbiamo eseguito due join con la tabella *Player*, prima recuperando i nomi dei loser e poi per i winner. Anche in questo caso le operazioni di giunzione sono state eseguite tramite il nodo *Ricerca*. A questo punto siamo in possesso di tutte le informazioni necessarie per rispondere alla domanda; per ogni perdente abbiamo il numero di sconfitte con ogni avversario con cui si è confrontato. La domanda ci chiede di restituire il nemesis di quel giocatore, ovvero il nome del giocatore con cui ha perso il maggior numero di partite.
Da notare che ogni player può avere 1 o più nemesis, infatti il maggior numero di partite perse possono occorrere con più di un avversario. Abbiamo notato che ci sono molti giocatori il cui massimo numero di sconfitte è pari ad 1 e di conseguenza hanno più nemesis. In genere questo accade nei confronti di giocatori emergenti, o comunque giocatori per i quali abbiamo pochi dati a disposizione, dove il numero complessivo di partite giocate è ridotto e il ventaglio di

avversari con cui si sono confrontati è ristretto.
Per selezionare soltanto il nemesis abbiamo suddiviso il flusso tramite il nodo *Multicast*. Nel primo flow abbiamo aggregato i dati raggruppando per *loser_id* e prendendo il massimo *num_defeats*. Il risultato del precedente flusso lo abbiamo mergiato con il secondo flusso (rimasto invariato), vincolando la join a considerare solo i perdenti con il massimo numero di partite perse. In entrambi i flussi, prima della merge, i dati sono stati ovviamente ordinati: crescentemente per il nome del giocatore sconfitto e descrescentemente per il numero di sconfitte. La tabella mergiata, dopo un ulteriore ordinamento, è stata salvata sul file *ass1_results* utilizzando il nodo *Destinazione OLE DB*.

Assignment 2

For each year, the name of the tournament with more matches played.

Abbiamo utilizzato il nodo *Origine OLE DB* per accedere alla tabella *Match*, filtrando soltanto gli attributi necessari per rispondere alla domanda, ovvero *match_id* e *tourney_id*. Successivamente, per reperire il nome del torneo e l'anno in cui è stato giocato, abbiamo eseguito due join, rispettivamente con le tabelle *Tournament* e *Date*, entrambe eseguite con un modulo *Ricerca*. A questo punto per contare i numeri di match distinti per ogni torneo abbiamo usato la funzione *Aggregazione*: raggruppando prima per anno, poi per nome del torneo e successivamente contando il numero di match nella nuova colonna: *num_matches*. A questo punto abbiamo tutte le informazioni per rispondere alla domanda: per ogni anno abbiamo i nomi dei tornei che sono stati giocati e il numero di matches che sono avvenuti. La domanda però ci chiede di restituire per ogni anno soltanto il nome del torneo composto dal maggior numero di matches. Per realizzare ciò abbiamo proceduto esattamente come nella business question precedente, ovvero con il nodo *Multicast*. Abbiamo quindi splittato i dati in due flussi: nel primo abbiamo aggregato i dati raggruppando per *tourney_name* e prendendo il massimo *num_matches*. Successivamente abbiamo mergiato il risultato del precedente flusso con il secondo, eseguendo la giunzione su *tourney_name* e il massimo *num_matches*. In questo modo la tabella risultante dalla merge contiene solo i nomi dei tornei con il massimo numero di matches (per anno). Ovviamente è stato necessario ordinare i dati prima di eseguire la *Merge join*. Infine, abbiamo salvato i dati sul file *ass2_results*, ordinati crescentemente per anno. Abbiamo scelto di aggregare per *tourney_name* e non per *tourney_id* perchè abbiamo notato che un torneo con lo stesso nome e giocato nello stesso anno, ha *tourney_id* differenti al variare del giorno in cui è stato giocato. Di conseguenza se avessimo raggruppato per *tourney_id* avremmo ottenuto il massimo numero di matches giocati in un torneo in una determinata data, e questo non è quello che avremmo voluto ottenere.

Parte 3

In questa parte abbiamo utilizzato SQL Analisis Services per costruire il datacube relativo al database creato nella prima parte e, successivamente, utilizzato MultiDimensional eXpressions in SQL Server Management Studio per rispondere ad alcune business questions. Infine è stato utilizzato Microsoft PowerBI per realizzare le dashboards richieste.

Assignment 0

Abbiamo costruito un datacube contenente i dati relativi alle tabelle dello schema precedentemente creato. In particolare, sono state create soltanto le dimensioni Tournament e Player che comprendono a loro volta Date e Geography. Questo è stato possibile creando le gerarchie CountryContienent e DayMonthQuarterYear all'interno delle due dimensioni. Inoltre, il cubo è stato costruito tenendo conto sia dello schema che delle business questions che ci sono state poste. Pertanto abbiamo utilizzato soltanto le measures e le gerarchie necessarie a rispondere alle queries. Ulteriori measures sono state realizzate appositamente per la realizzazione delle dashboards.

Innanzi tutto abbiamo stabilito una connessione col database contenente le tabelle. Al fine di rendere più comprensibile la gerarchia temporale contenuta in Date, abbiamo derivato e aggiunto gli attributi the_month e the_quarter; i quali contengono i riferimenti letterali ai mesi e ai quadrimestri (ie. 'February','Q2') di ogni anno.

Dopodichè sono state definite le dimensioni del cubo: Winner, Loser e Tournament. Le prime due contengono tutti gli attributi della tabella Player e la gerarchia CountryContinent, con la dipendenza funzionale da country_ioc verso continent. La dimensione Tournament contiene oltre agli attributi dell'omonima tabella, la gerarchia DayMonthQuarterYear la quale utilizza le colonne precedentemente derivate e definisce le dipendenze da the_month verso month e da the_quarter verso quarter. Le dipendenze funzionali oltre a definire la gerarchia ci hanno permesso di garantire l'ordinamento lessicografico degli stessi.

Infine abbiamo creato il cubo, utilizzando soltanto le measures necessarie a soddisfare le business questions. Esse sono: winner_rank_points, loser_rank_points, winner_rank, loser_rank e N_winners. Quest'ultima measure è stata creata per rispondere al primo assignment ed è viene utilizzata per contare il numero totale di vincitori di un insieme di match (univocamente). Per la realizzazione della dashboard a piacere abbiamo considerato ulteriori measures. Dopo aver settato le funzioni di aggregazione di ogni measures, abbiamo processato l'OLAP cube sui server del dipartimento così da poterlo successivamente interrogare.

Assignment 1

Show the total winners for each country and the grand total with respect to the continent.

In questa query ci viene chiesto di mostrare il totale dei vincitori per ogni country e il totale complessivo rispetto al continente di quel paese. Il numero dei vincitori è definito nella measure N_winners, che abbiamo opportunamente creato in fase di costruzione del cubo. La funzione di aggregazione di N_Winners è stata settata su DISTINCT COUNT così da contare in modo univoco i vincitori. Abbiamo scritto la query MDX riportando sulla prima colonna il totale dei vincitori rispetto al paese mentre nella seconda il totale rispetto al rispettivo continente. In ogni riga oltre al paese abbiamo scelto di visualizzare anche il nome del continente in modo tale da facilitare la lettura ed avere un immediato riscontro rispetto alla sigla del paese. Non potendo accedere contemporaneamente a due differenti livelli della gerarchia, abbiamo eseguito il crossjoin dei flat attributes (continent e country_ioc).

Assignment 1 bis

Show the total winners for each country and the radio wrt the grand total to the continent.

Abbiamo sviluppato una variante alla stessa query considerando, invece che il grand total, il rapporto tra il numero di vincitori di quel paese e il totale del continente. In questo modo siamo in grado di verificare quando un paese incide sul suo continente, in termini di numero di vincitori.

Assignment 2

Show the total winner rank points for each year and the running yearly winner rank points for European players.

In questa seconda query ci viene chiesto di mostrare i winner rank points totali per ogni anno e quelli cumulati annualmente fino all'anno considerato, il tutto considerando soltanto players europei. La query è stata scritta riportando nella prima colonna i winner rank points e nella seconda i running yearly rank points, mentre ogni riga riporta l'anno. I running yarly rank points sono stati con un mebro derivato tramite lo slicing, in particolare considerando tutti i winner rank points degli anni che intercorrono tra il 2016 (primo anno disponibile nei dati) e il currentmember (year). Abbiamo utilizzato la metrica aggregate in quanto in fase di costruzione del cubo la funzione di aggregazione di winner rank points è stata impostata su sum. Infine, i player europei sono stati filtrati con la clausola where.

Assignment 3

Show the ratio between the total winner rank points of each year w.r.t the previous year.

Con quest'ultima query ci viene chiesto di mostrare il rapporto tra il totale dei winner rank points di ogni anno rispetto al precedente. Anche in questo caso, come nella query precedente, ogni riga riporta l'anno, la prima colonna i winner rank points mentre la seconda il ratio. Quest'ultimo lo abbiamo calcolato rapportando i winner rank points dell'anno corrente rispetto ai winner rank points dell'anno precedente ottenuto con la clausola lag(1), la quale prende in considerazione l'anno precedente a quello corrente. Ovviamente il ratio del primo anno (2016) vale infinito in quanto non sono presenti dati relativi ad anni precedenti. Il formato, infine, è stato settato su percent.

Assignment 4

Abbiamo utilizzato Microsoft Power BI per creare le dashboards. Per mostrare la distribuzione geografica dei winner rank points e loser rank points è stata usata la "Mappa". I valori delle distribuzioni possono essere visualizzati passando sopra le bolle del paese desiderato. Nella legenda abbiamo inserito la gerarchia in base all'anno, così da visualizzare le informazioni in base all'anno. Le bolle assumono la forma di grafico a torta.

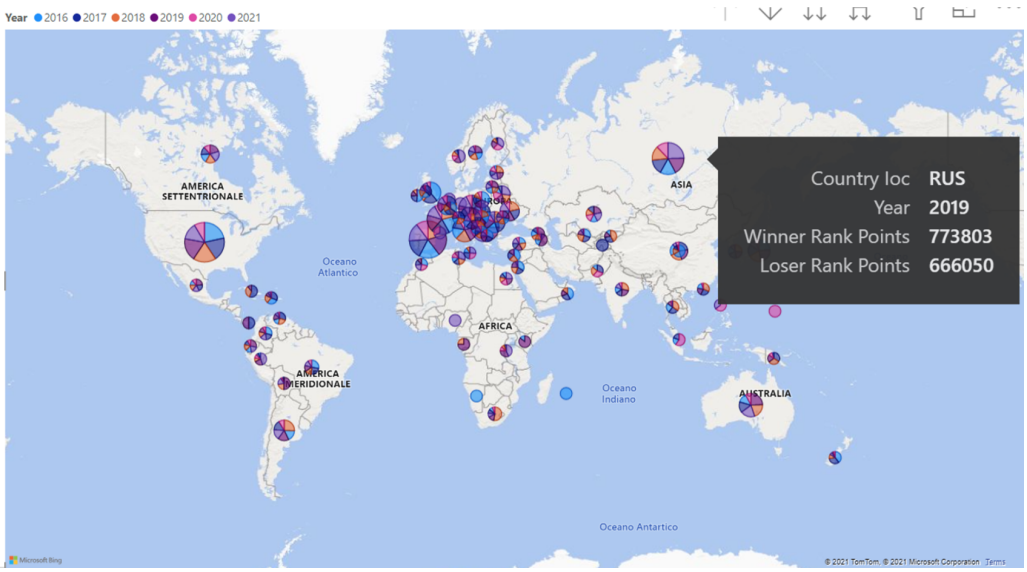

Figure 3: Distribuzione geografica di Winner/Loser Rank Points in funzione dell'anno.

Assignment 5

Abbiamo utilizzato un "Grafico a Torta" per visualizzare il numero di matches giocati negli anni. A differenza della dashboard precedente, abbiamo utilizzato tutta la gerarchia temporale in modo tale da visualizzare con maggior dettaglio il numero di partite giocate in ogni periodo dell'anno (ie. quadrimestre, mese, giorno). Inoltre, abbiamo riportato come informazione aggiuntiva il numero di ace totali messi a segno da ogni player di ogni match. Le measures aggiuntive (w_ace, l_ace e N_Winners) sono state aggiunte nel cubo OLAP appositamente per la realizzazione di questa dashboard, modificandone opportunamente le funzioni di aggregazione.

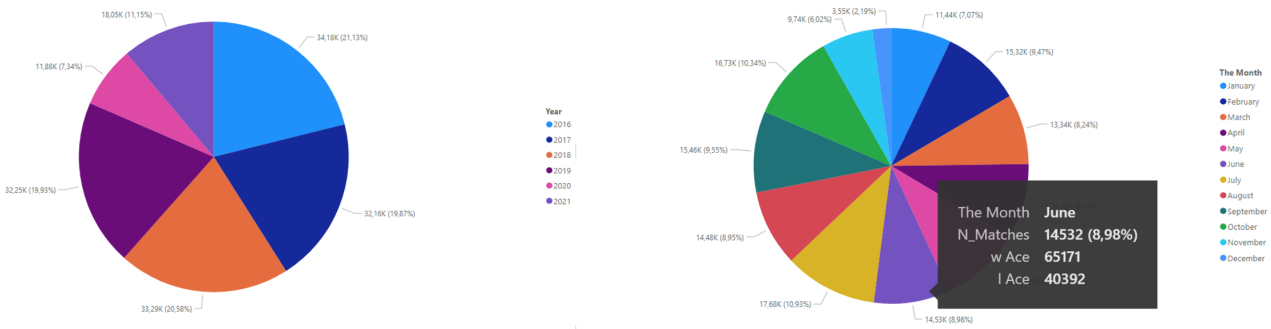

Figure 4: A sinistra il numero di matches per anno mentre a destra abbiamo effettuato 2 drill down visualizzando le distribuzioni sui mesi.