Qualificazioni

alla Soccer Data Challenge 2019

Dati a disposizione

Il dati da analizzare sono disponibili a questo link: **http://bit.ly/dataset_qualificazioni**.
Scaricate il file .zip, nel quale troverete quattro documenti json:

- `matches.json`
- `teams.json`
- `players.json`
- `events.json`

La documentazione con i dettagli per interpretare il contenuto dei file è disponibile al link **https://apidocs.wyscout.com/**

Partita da analizzare

La partita da analizzare è **Internazionale - Juventus** del 28 Aprile 2018. È la partita che, nel file `matches.json`, ha il campo `wyId` uguale a `2576302`. Gli eventi relativi a questa partita sono quelli che, nel documento `events.json`, hanno il campo `matchId` uguale a `2576302`.

File da consegnare

Per ogni esercizio, i partecipanti devono caricare sull'apposito form:
http://bit.ly/form_qualifiche_2019

1. i risultati in un file di testo in formato `.csv` (comma separated values), Il separatore da usare è il punto e virgola "`;`". Nel file `.csv` usare "`\n`" come separatore delle righe.

2. il file `.py` con il codice Python 3 usato per creare il file `.csv`.

Da notare bene:

- Il nome del file `.csv` e del file `.py` devono essere **esattamente** quelli indicati nella traccia degli esercizi sottostanti. Gli esercizi associati a file con nomi diversi da quelli indicati saranno ritenuti errati, indipendentemente dal file `.csv` prodotto.

- La struttura del file `.csv` consegnato, incluso il nome e l'ordine delle colonne, deve essere **esattamente** quello indicato come "header" nella traccia degli esercizi. Gli esercizi associati a file `.csv` con nomi diversi di colonne o ordine di colonne diverso da quelli indicati saranno ritenuti errati, indipendentemente dal file `.csv` prodotto.

- I risultati nel file `.csv` devono essere ordinati in base a quanto eventualmente richiesto nella traccia degli esercizi. Gli esercizi associati a file `.csv` i cui risultati sono ordinati in modo diverso da quanto specificato saranno ritenuti errati, indipendentemente dal file `.csv` prodotto.

- **Non è possibile utilizzare nessuna libreria** (in altre parole pandas, numpy, scikit-learn, matplotlib, ecc., non sono consentite). L'unica libreria consentita è la libreria `json` (https://docs.python.org/3.5/library/json.html), che serve per caricare i file con i dati da analizzare. In altre parole, l'unico import consentito è `import json`. Gli esercizi che utilizzano librerie ausiliarie saranno considerati errati indipendentemente dal file `.csv` prodotto.

Modalità di correzione

Gli esercizi saranno corretti in modo automatico da un algoritmo che verificherà che il file `.csv` prodotto dal file `.py` sia corretto. L'algoritmo che eseguirà la verifica automatica assume che la versione di Python utilizzata sia la 3.0 o versioni successive e che i quattro file del dataset (`matches.json`, `teams.json`, `players.json`, `events.json`) siano nella stessa cartella dei file .py.
Notate bene: il codice deve poter essere eseguito su qualsiasi computer, è quindi importante non usare i path assoluti nel codice fornito (esempio: C:\Users\Utente\Challenge\events.json). Gli esercizi vengono corretti in modo automatico assumendo che i file caricati si trovino nella stessa cartella del file `.py`.

Parametri valutazione

1. Numero di esercizi svolti correttamente
2. 1 punto in più assegnato per chi ha una squadra bilanciata tra componente maschile e femminile come definito nel regolamento
3. Tempo in secondi rimasto alla conclusione delle qualifiche

Esercizi

Verrà assegnato un punto per ogni esercizio svolto correttamente, ossia per ogni esercizio che 1) il file Python produce esattamente il file `.csv` caricato e 2) il `.csv` prodotto è corretto sia nella formattazione che nei risultati contenuti.

1. Conta il numero di eventi di ogni giocatore della squadra di casa. Ricordati di considerare anche i giocatori subentrati e sostituiti e di escludere gli arbitri (che hanno `playerId` uguale a `0`). Ordina i risultati in modo crescente per `playerId`.

    a. Nome file:
    `esercizio1.csv`
    `esercizio1.py`

    b. header:
    `playerId;n_events`

2. Conta il numero di eventi per ogni squadra. Ricordati di considerare anche i giocatori subentrati e sostituiti e di escludere gli arbitri. Ordina i risultati in modo crescente per `teamId`.

    a. Nome file:
    `esercizio2.csv`
    `esercizio2.py`

    b. header:
    `teamId;n_events`

3. Conta i passaggi di ogni giocatore della squadra di casa. Nota che un passaggio è identificato dalla presenza della stringa `Pass` nel campo `eventName`. Ricordati di considerare anche i giocatori subentrati e sostituiti e di escludere gli arbitri. Ordina i giocatori in modo crescente per `playerId`.

    a. Nome file:
    `esercizio3.csv`
    `esercizio3.py`

    b. header:
    `playerId;n_pass`

4. Conta gli eventi `Simple pass` di ogni giocatore della squadra fuori casa. Nota che `Simple pass` è un "sottoevento", quindi bisogna accedere al campo `subEventName` di un evento per trovarlo. Ricordati di considerare anche i giocatori subentrati e sostituiti e di escludere gli arbitri. Ordina i giocatori in modo decrescente per `playerId`.

    a. Nome file:
    `esercizio4.csv`
    `esercizio4.py`

    b. header:
    `playerId;n_simple_pass`

5. Conta il numero dei falli subiti dalla squadra di casa nel secondo tempo. Nota che l'evento `Foul` si riferisce ai falli commessi dalla squadra corrispondente al `teamId` dell'evento. Il tempo di gioco è indicato dal campo `matchPeriod`.

    a. Nome file:
    `esercizio5.csv`
    `esercizio5.py`

    b. header:
    `n_falli_subiti`

6. Conta il numero di falli commessi nel primo tempo da ogni giocatore della squadra fuori casa. Ricordati di considerare anche i giocatori subentrati e di escludere gli arbitri. Ordina i risultati in modo decrescente per `playerId`.

    a. Nome file:
    `esercizio6.csv`
    `esercizio6.py`

    b. header:
    `playerId;n_falli_commessi`

7. Conta i dribbling riusciti effettuati da ogni attaccante a partire dal 20° minuto del secondo tempo compreso (ossia a partire dal secondo 1200 compreso). Il dribbling è identificato dal sottoevento di nome `Ground attacking duel`,che si trova nel campo `subEventName`. Il momento in cui avviene un evento è indicato dal campo `eventSec` (espresso in secondi a partire dall'inizio del tempo di gioco in corso). La presenza del tag `1801` indica che un evento è accurato (nel caso del dribbling quindi che è riuscito). Gli attaccanti sono i giocatori di ruolo `Forward` (campo `role` e sottocampo `name`) nel file `players.json`. Ordina i risultati in modo crescente per `playerId`.

    a. Nome file:
    `esercizio7.csv`
    `esercizio7.py`

    b. header:
    `playerId;n_dribbling`

8. Conta i tiri effettuati subito dopo un dribbling vinto. Il dribbling è identificato dal sottoevento di nome `Ground attacking duel`, che si trova nel campo `subEventName`. Quando l'evento successivo a `Ground attacking duel` è un evento `Ground defending duel`, non considerarlo come evento immediatamente successivo.[1] La presenza del tag `1801` indica che un evento è accurato (nel caso del dribbling quindi che è stato vinto).

    a. Nome file:
    `esercizio8.csv`
    `esercizio8.py`

    b. header:
    `n_dribbling`

9. Calcola il numero massimo di passaggi (`Pass` nel campo `eventName`) accurati consecutivi di entrambe le squadre. Ricorda che un passaggio accurato è indicato dalla presenza del tag `1801`. Ordina i risultati in modo crescente per `teamId`.

    a. Nome file:
    `esercizio9.csv`
    `esercizio9.py`

    b. header:
    `teamId;n_pass_azione`

[1] Spesso i duelli appaiono in sequenza: per un ground attacking duel della squadra A c'è un ground defending duel della squadra B. Questo accade perché, appunto, quando un giocatore tenta il dribbling c'è sicuramente un giocatore avversario che subisce il dribbling. L'ordine dei due eventi, però, dipende da quale dei due ha "iniziato" il duello. Perciò può capitare di incontrare queste sequenze: ground defending-ground attacking-shot oppure ground attacking-ground defending-shot. Entrambi i tiri vanno considerati come successivi ad un dribbling.

10. Calcola la distanza media dei tiri rispetto al centro della porta (che è in posizione x=100, y=50) effettuati dalla squadra di casa. Nel caso in cui la distanza media calcolata risulti con più di 4 cifre decimali, arrotonda alla quarta cifra decimale (usare la funzione `round`).
    Suggerimento: usare la distanza euclidea (ricorda che non è possibile importare nessuna libreria che non sia la libreria `json`)

    a. Nome file:
       `esercizio10.csv`
       `esercizio10.py`

    b. header:
       `distanza_media`

11. Calcola la posizione media dei tackle effettuati dalla squadra fuori casa. Il tackle è identificato dal sottoevento `Ground defending duel`, che si trova nel campo `subEventName`. Nel caso in cui le coordinate della posizione media calcolate risultino con più di quattro cifre decimali, arrotonda alla quarta cifra decimale. Come posizione del tackle usa il punto in cui il tackle comincia. Nota che la posizione di un evento è specificata nel campo `positions` come una lista di dizionari e che il punto dove inizia l'evento è il primo di questi dizionari.

    a. Nome file:
       `esercizio11.csv`
       `esercizio11.py`

    b. header:
       `p_tackle_x;p_tackle_y`

12. Conta i calci d'angolo battuti dai giocatori subentrati a partita iniziata. I calci d'angolo sono definiti dal valore `Corner` nel campo `subEventName`. I giocatori subentrati a partita iniziata sono elencati nel campo `teamsData` (sottocampi `formation` e `substitutions`) del file `matches.json`. Ordina i risultati in modo decrescente per `playerId`.

    a. Nome file:
    `esercizio12.csv`
    `esercizio12.py`

    b. header:
    `playerId;corner_kicks`

13. Calcola il rapporto tra il numero di falli commessi e il numero di cartellini gialli subiti da ognuna delle due squadre. Il cartellino giallo è indicato dalla presenza del tag `1702`. Nel caso in cui il rapporto calcolato risulti con più di 4 cifre decimali, arrotonda alla quarta cifra decimale. Ordina i risultati in modo crescente per `teamId`.

    a. Nome file:
    `esercizio13.csv`
    `esercizio13.py`

    b. header:
    `teamId;ratio`

14. Calcola il tempo medio tra due eventi consecutivi durante la partita. Nota che il tempo tra l'ultimo evento del primo tempo e il primo evento del secondo tempo non va considerato. Nel caso in cui la media calcolata risulti con più di quattro cifre decimali, arrotonda alla quarta cifra decimale.

    a. Nome file:
    `esercizio14.csv`
    `esercizio14.py`

    b. header:
    `t_btw_eventi`

15. Calcola per ogni tiro la distanza sull'asse delle x tra il punto dove inizia l'evento e la porta avversaria. Ricorda che il valore della coordinata x della porta avversaria è 100. Nota che la posizione dell'evento è specificata come una lista di dizionari e che il punto dove inizia l'evento è il primo di questi dizionari. Ordina i tiri in modo decrescente per tempo di gioco (prima il secondo tempo e poi il primo tempo) e a parità di tempo di gioco per momento di gioco (campo `eventSec`).

    a. Nome file:

    `esercizio15.csv`
    `esercizio15.py`

    b. header:

`tempo;momento;distanza`

16. Indica il nome della nazionalità e il `playerId` dei 3 giocatori della squadra di casa che hanno effettuato più passaggi sbagliati. Nota che un passaggio sbagliato è indicato dalla presenza del tag `1802` negli eventi `Pass`. Il nome della nazionalità di un giocatore è indicato nel campo `birthArea` e nel sottocampo `name` del file `players.json`. Il `playerId` nel file `players.json` è indicato dal campo `wyId`. Ordina i risultati in modo crescente per `playerId`.

    a. Nome file:
    `esercizio16.csv`
    `esercizio16.py`

    b. header:
    `playerId;nationality;n_pass_sbagliati`

17. Indica il nome del ruolo e il `playerId` dei 3 giocatori della squadra fuori casa che hanno intercettato più volte la palla. Il nome del ruolo del giocatore è indicato nel campo `role` e nel sottocampo `name` del file `players.json`. Nota che l'intercettazione di palla è indicata dalla presenta dal tag `1401` in un qualsiasi evento. Ordina i risultati in modo decrescente per `playerId`.
    Suggerimento: il giocatore con `playerId` uguale a 0 è l'arbitro e va escluso.

    a. Nome file:
    `esercizio17.csv`
    `esercizio17.py`

    b. header:
    `playerId;role;n_intercetti`

18. Conta i passaggi accurati effettuati dal giocatore Higuaìn (`playerId`=3323) dopo il 30° del primo tempo compreso (1800 secondi dall'inizio della partita) che iniziano nella metà campo avversaria. Nota che la posizione di un evento è specificata come una lista di dizionari e che il punto dove inizia un passaggio è il primo di questi dizionari. La metà campo avversaria è specificata dalle coordinate [50, 100] dell'asse x (50 e 100 compresi). Un passaggio accurato è indicato dalla presenza del tag `1801` negli eventi `Pass`.

    a. Nome file:
    `esercizio18.csv`
    `esercizio18.py`

    b. header:
    `n_pass_accurati`

19. Indica il `playerId` e il peso dei 3 giocatori che hanno effettuato più duelli aerei nella fascia centrale del campo (posizione dalla 25 alla 75 comprese sull'asse delle y). I duelli aerei sono indicati dall'evento `Duel` nel campo `eventName` e dal sottoevento `Air duel` nel campo `subEventName`. Nota che il peso di un giocatore è indicato dal campo `weight` del file `players.json`. Per la posizione del duello, considerare soltanto dove l'evento inizia. Ordina i risultati in modo decrescente per `playerId`.
    Suggerimento: il giocatore con `playerId` uguale a 0 è l'arbitro e va escluso.

    a. Nome file:
       `esercizio19.csv`
       `esercizio19.py`

    b. header:
       `playerId;weight;n_duelli_aerei`

20. Conta le parate effettuate da ogni portiere. Una parata è indicata da `Save attempt` nel campo `eventName`. Ordina i risultati in modo crescente per `n_parate`. A parità di numero di parate, ordina in modo crescente per `playerId`.

    a. Nome file:
       `esercizio20.csv`
       `esercizio20.py`

    b. header:
       `playerId;n_parate`

21. Per ogni giocatore conta quanti tiri ha effettuato in porta in alto a destra (presenza del tag `1209`), fuori dalla porta in basso a destra (presenza del tag `1210`) e quanti pali a sinistra ha colpito (presenza del tag `1218`). Ordina i risultati in modo crescente per `n_tiri`, a parità di `n_tiri` ordina in modo crescente per `playerId`, a parità di `playerId`, ordina in modo crescente per `tipo_tiro`. Come valori della colonna `tipo_tiro` usa i tag corrispondenti (ossia `1209`, `1210`, `1218`). Nota che un giocatore può apparire più volte nel file risultante.

    a. Nome file:
    `esercizio21.csv`
    `esercizio21.py`

    b. header:
    `tipo_tiro;playerId;n_tiri`

22. Conta le spazzate effettuate dalla trequarti difensiva (posizione da 0 a 25 compresi sull'asse delle x). Le spazzate sono indicate dal sottoevento `Clearance` nel campo `subEventName`.

    a. Nome file:
    `esercizio22.csv`
    `esercizio22.py`

    b. header:
    `n_spazzate`

23. Per il primo tempo e il secondo tempo della partita separatamente, conta il numero di eventi per ogni slot di 10 minuti. Gli slot devono essere progressivi partendo dal numero 0. Raggruppa i minuti di recupero insieme allo slot che parte dal 40° minuto. Gli slot del secondo tempo devono ripartire dalla numerazione 0. La colonna `tempo` può assumere valori `1H` (che indica il primo tempo) oppure `2H` (che indica il secondo tempo). Ordina in modo crescente per tempo (ossia prima gli `1H` e poi i `2H`), a parità ordina in modo crescente per numero di slot.
    Suggerimento: il primo slot contiene gli `eventSec` nel range [0, 600) (ossia 0 compreso e 600 escluso), il secondo slot contiene gli `eventSec` nel range [600, 1200) (ossia 600 compreso e 1200 escluso), ecc.

    a. Nome file:
    `esercizio23.csv`
    `esercizio23.py`

    b. header
    `slot;tempo;n_events`

    c. esempio di file `.csv` di output (i valori sono della colonna `n_events` sono puramente indicativi):

```
slot;tempo;n_events
0;primo;12
1;primo:16
2;primo;20
3;primo;18
4;primo:8
0;secondo;22
1;secondo:12
2;secondo;28
3;secondo;16
4;secondo:8
```