

Cloud, Distributed and Parallel Computing

Prof. Anisetti, Ardagna, Gaudenzi

Progetto di

Sabrina Cenghialta, Laura Romano, Guido Ravasi, Matteo Serra

Task progetto di Analisi Dati

Il progetto di esame deve comprendere le seguenti componenti: Relazione di progetto (che riporta gli snippet di codice commentato e descrive come le scelte di sviluppo si rifanno ai 12Factor, manuale di build e deployment, Repository Git con codice, script di installazione e dockerfile).

I progetti disponibili per Analisi dati sono rintracciabili:

- <https://github.com/awesomedata/awesome-public-dataset>
- <https://github.com/italia/awesome-italian-public-datasets>

Dataset e Repository

Nel nostro caso, abbiamo svolto un progetto utilizzando il dataset “FIFA 2021 Complete Player Dataset”, pubblicato su Kaggle:

[FIFA 2021 Complete Player Dataset | Kaggle](#)

Il progetto è presente sul seguente repository:

<https://github.com/MrTeoTZR/Cloud-project-Fifa>

Twelve Factor

Considerato che nel corso e nella richiesta del progetto viene posta enfasi ai “Twelve Factor”, riteniamo opportuno effettuare una breve descrizione delle 12 caratteristiche che un’applicazione Cloud Native deve avere e che terremo rigorosamente presenti nello svolgimento del progetto:

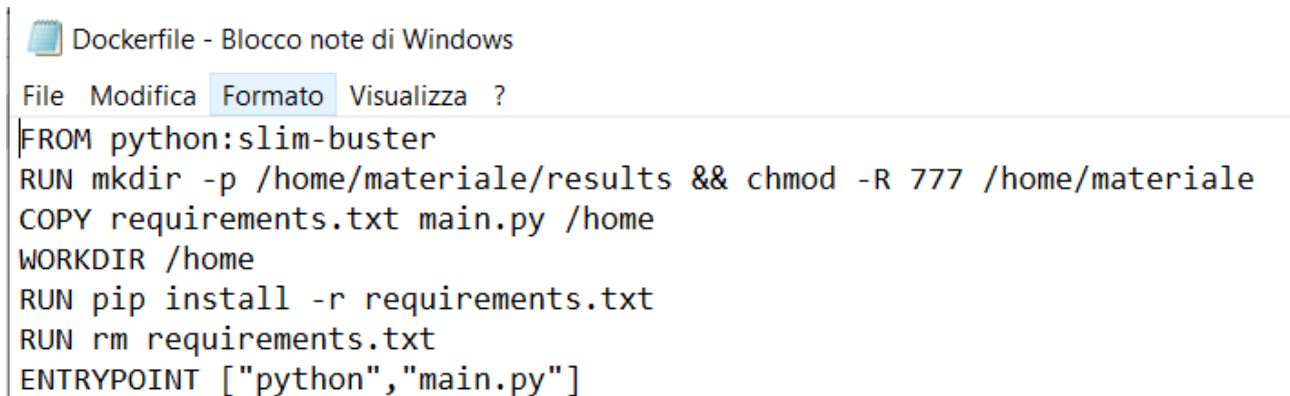
1. *Codebase*: è l’insieme di tutto il codice necessario per l’app (codice di installazione, codice di configurazione, codice di promozione ecc). Il codice deve essere versionato e deve poter gestire tutti i deploy di cui abbiamo bisogno.
2. *Dependencies*: occorre poter gestire in maniera opportuna le dipendenze, quindi dovremo esplicitamente dichiarare le dipendenze (e le loro versioni). Le dipendenze sono tutte le librerie necessarie all’applicazione per poter funzionare. Ogni microservizio isola e impacchetta le proprie dipendenze, includendo cambiamenti senza influenzare l’intero sistema.

3. *Configuration*: Nel codebase deve essere salvato tutto quello che riguarda la configurazione. File di configurazione significa che l'applicazione deve poter essere compatibile con diversi ambienti. Le informazioni di configurazione vengono rimosse dai microservizi ed esternalizzate al di fuori del codice, attraverso uno strumento di gestione della configurazione.
4. *Backing Services*: I backing services sono quei servizi dove appoggiamo le informazioni. Sono servizi che ci rendono l'applicazione statefull (come i database, le code, il file system, gli object storage come AWS S3). Sono i luoghi in cui la nostra applicazione va a prendere informazioni. Dobbiamo gestire i backing services come risorse disaccoppiate dall'applicazione, che possono essere modificate attraverso le configuration.
5. *Build, Release, Run*: Build, release e run devono essere separate, quindi occorre riuscire a costruire una catena dove l'operazione di build, l'operazione di release e l'operazione di run risultino essere 3 operazioni ben distinte.
6. *Processes*: Ogni microservizio dovrebbe essere eseguito nel proprio processo, isolato da altri servizi in esecuzione. L'applicazione che sviluppiamo deve avere uno stato [un salvataggio] che si trova fra i backing services. Ogni macchina deve essere stateless: deve salvare lo stato da un'altra parte.
Lo step 6 è molto importante perché l'applicazione Cloud Native deve poter scalare orizzontalmente.
7. *Port Binding*: occorre sapere quali porte vengono esportate dal mio servizio. Il servizio ha un indirizzo, delle porte e delle route e occorre sapere quale porta viene esportata.
8. *Concurrency*: questo fattore esamina le pratiche per il ridimensionamento dell'app. Tali pratiche vengono utilizzate per gestire ciascun processo nell'app in modo indipendente (ad esempio avvio / arresto, clonazione su macchine diverse ecc). Il fattore si occupa anche di suddividere l'app in pezzi molto più piccoli. La concurrency prevede anche che ci sia un modello che scali gestendo la concorrenza in maniera opportuna.
9. *Disposability*: la disposability esamina la robustezza dell'app con metodi di avvio e spegnimento rapidi. Poiché vogliamo essere in grado di scalare orizzontalmente (quindi aggiungere risorse e toglierle), l'applicazione deve poter avere nuove risorse in pochissimo tempo e, quando tolgo risorse, non devo creare problemi all'infrastruttura. Le applicazioni Cloud sono applicazioni nate per gestire il fallimento: devono poter gestire il fatto che non ci sia il backing service o che ci siano picchi di richieste inaspettate. Devono quindi essere app robuste, sviluppate con in mente la security by design (sviluppate pensando agli aspetti di sicurezza).
10. *Development and Production Parity*: riprende i primi 4 punti: le dipendenze non devono essere costruite per l'ambiente di development e per l'ambiente di sviluppo ma ci vuole un solo file di dipendenze che sia efficiente sia per la produzione, sia per lo sviluppo. In caso di distribuzione continua, è necessario disporre di un'integrazione continua basata su ambienti corrispondenti per limitare deviazioni ed errori.
11. *Registri*: dobbiamo poter fare monitoraggio di ciò che succede. I meccanismi di registrazione sono fondamentali per il debug. I logs (file dove vengono tracciate le operazioni) devono essere gestiti come flussi di eventi facilmente riportabili per fare analisi.
12. *Admin Processes*: tutte le operazioni di amministrazione dell'applicazione (e quindi tutte le operazioni di configurazione e set up) devono essere fatte in un colpo, altrimenti si rischiano problemi di sincronizzazione.

Contenuto del repository e descrizione delle attività svolte

Nel repository indicato sono presenti diversi file: il DockerFile, il file Requirement.txt, il file Python main.py oltre al ReadMe, che andremo a presentare puntualmente per illustrare il lavoro svolto.

DockerFile



```
FROM python:slim-buster
RUN mkdir -p /home/materiale/results && chmod -R 777 /home/materiale
COPY requirements.txt main.py /home
WORKDIR /home
RUN pip install -r requirements.txt
RUN rm requirements.txt
ENTRYPOINT ["python", "main.py"]
```

Figura 1 – Dockerfile

Il file Dockerfile è il file che viene richiamato durante la fase di build per la creazione del container. Quando viene richiamato, va a costruire il container Docker. Una volta che il container è pronto, si va poi a far partire il progetto in Python. Vediamo cosa fa il Docker file nello specifico osservando le righe contenute nel documento riprodotto nella Figura sopra.

Con la prima riga *FROM python: slim-buster* andiamo a creare una immagine docker a partire dalla slim-buster offerta su Docker Hub. Si tratta di una copia di una macchina slim-buster che permette di accedere ad un sistema operativo ubuntu. La macchina è però al momento vuota e dobbiamo inserire nel container tutti gli elementi necessari per far partire il nostro programma di analisi.

Con la seconda riga *RUN mkdir -p /home/materiale/results && chmod -R 777 /home/materiale* andiamo a creare con il comando con mkdir sul Docker le directory seguenti: home/materiale e home/materiale/results. Con la istruzione chmod – R 777, invece, stiamo dando i permessi di lettura, scrittura ed esecuzione su queste cartelle.

Nella terza riga di codice *COPY requirements.txt main.py /home* copiamo il file requirements e il main all’interno della cartella virtuale “home”.

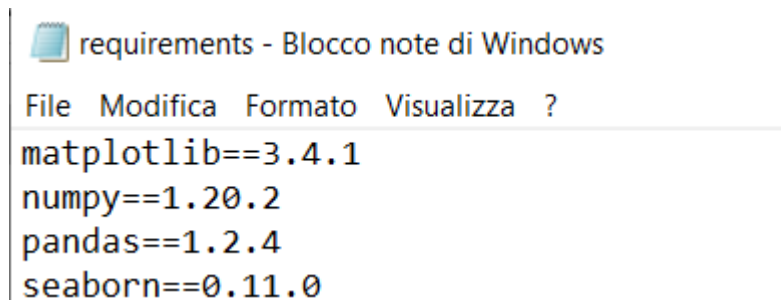
Quindi, nella quarta riga *WORKDIR /home*, settiamo come working director ossia directory di lavoro principale la cartellina home. Questo comporta che tutte le istruzioni che lanceremo da qui in poi saranno inserite automaticamente in questa cartella.

Con il comando *RUN pip install -r requirements.txt*, che costituisce il contenuto della quinta riga di codice, installiamo tutte le librerie di python che sono presenti nel file requirements.txt.

Nella sesta riga, con il *RUN rm requirements.txt* facciamo la remove del file requirements.txt. In altri termini se poco prima era stato copiato all’interno della home ora procediamo alla sua rimozione.

Abbiamo fatto la build del docker, lo abbiamo costruito e adesso dobbiamo indicare cosa utilizzare per far partire con run. Nell'ultima riga `ENTRYPOINT ["python", "main.py"]` indichiamo cosa far partire al run nel docker: python e il main. Abbiamo visto, nella disamina de 12 factor, che build e run devono essere separati.

Requirements.txt



```
requirements - Blocco note di Windows
File  Modifica  Formato  Visualizza  ?
matplotlib==3.4.1
numpy==1.20.2
pandas==1.2.4
seaborn==0.11.0
```

Figura 2 – file requirements.txt

Inseriamo in questo file le *librerie* di cui abbiamo bisogno. Nel nostro caso inseriamo: matplotlib, numpy, pandas e seaborn.

Si tratta, come è noto, di quattro librerie essenziali. Dopo il nome della libreria, indichiamo, come da buona pratica, la *versione* utilizzata della libreria corrispondente. Questo va sottolineato perché *indicare la versione della libreria non è opzionale*, ma ci viene richiesto dal fattore delle dipendenze. Secondo questo fattore occorre esplicitamente dichiarare le dipendenze e le loro versioni per non pregiudicarci il funzionamento nel caso incontrassimo delle versioni diverse, superate o comunque non congrue, dove non sono presenti funzioni invece contemplate in versioni successive e più aggiornate.

Main.py

Il file main.py è un file Python, creato tramite PyCharm, che permette al docker di effettuare una determinata data analysis del dataset sui calciatori di Fifa 2021. Andiamo ad analizzarne i passi principali, in modo da comprenderne il funzionamento.

La parte iniziale è dedicata all'import delle librerie necessarie per l'esecuzione dello script. Subito dopo, abbiamo definito la cartella dove verranno salvati i risultati delle analisi attraverso la creazione della variabile "dir_out".

```
dir = str(pathlib.Path().parent.absolute())
dir_out = "{}materiale/results/".format(dir)
```

Segue la definizione di una funzione avente lo scopo di controllare l'eventuale esistenza della cartella necessaria per il salvataggio degli output. Nel caso in cui la cartella non esistesse, la funzione stessa fornisce le istruzioni per la creazione della directory.

```
def esistenza_directory(dir):
    if os.path.isdir(dir) == False:
        logging.warning("Directory di output: {} non esistente".format(dir))
        try:
            #Se non esiste la creo
            os.makedirs(dir)
            logging.info("Creazione directory: {} ".format(dir))

        except OSError as error:
            logging.error("Errore durante la creazione della directory: {}".format(dir))

    else:
        logging.info("Directory di output: {} esistente".format(dir))
```

Lo script prosegue istanziando il logger, che servirà per la creazione nella directory corrente del file di log, chiamato "filelog.log". Il logger sarà in grado di restituire messaggi di tipo informativo, d'errore o di warning.

```
logging.basicConfig(handlers=[logging.FileHandler(filename='filelog.log', encoding='utf-8', mode='a+')],format='%(asctime)s - %(levelname)s - %(message)s', level=logging.INFO)
logger = logging.getLogger()
logging.info("RUN docker with parameter: {} ".format(sys.argv[1] if len(sys.argv)>1 else 'NULL' ))
```

Al fine di porre in essere le condizioni necessarie per effettuare l'analisi di un dataset, abbiamo impostato il caricamento del file "FIFA2021.csv" in modo che lo script sia autonomamente in grado di scaricarlo dal link inserito per poi trasformarlo in un dataframe Pandas.

```
percorso_remoto = 'https://github.com/MrTeoTZR/Cloud-project-Fifa/raw/master/FIFA2021.csv'
try:
    df = pd.read_csv(percorso_remoto, sep = ';')
    logger.info("Lettura dataset")
except Exception as e:
    logger.error("Non è stato possibile leggere il file all'indirizzo: {}".format(percorso_remoto))
```

Una volta caricato il dataframe, abbiamo creato 14 funzioni che permettono di effettuare 14 diverse analisi.

```
def giocatori_nazione():
    plt.figure(figsize = (20,7))
    df['nationality'].value_counts().head(10).plot.bar(color = sns.color_palette('viridis'))
    plt.title('Giocatori per nazione in FIFA')
    plt.xlabel('Nazionalità')
    plt.ylabel('Conteggio')
    filename = "{}.png".format('Giocatori per nazione in FIFA-2021')
    plt.savefig(dir_out+filename,bbox_inches='tight',dpi=500,transparent=True)
```

Queste analisi possono essere effettuate direttamente dal container, aggiungendo il tipo d'analisi preferito nel comando di run. A seconda del parametro scelto, si otterranno output differenti, con analisi specifiche sui dati.

Nello script sotto indichiamo tutte le opzioni in codice e le loro descrizioni.

```
opzioni=["-opzioni",
        "-gn",
        "-ng",
        "-sp",
        "-gg",
        "-gv",
        "-sg",
        "-sv",
        "-gt",
        "-di",
        "-at",
        "-ce",
        "-as",
        "-ad",
        "-po",
        "-all"]
descr_opzioni=[ "Tutte le opzioni",
                "Giocatori per nazione in FIFA-2021",
                "Età dei giocatori in FIFA",
                "Squadre di club più popolari in FIFA-2021",
                "Giocatori più giovani in FIFA.png",
                "Giocatori più vecchi in FIFA.png",
                "Squadre mediamente più giovani in FIFA.png",
                "Squadre mediamente più vecchie in FIFA.png",
                "Giocatori più forti in FIFA.png",
                "Difensori centrali più forti in FIFA.png",
                "Attaccanti più forti in FIFA.png",
                "Centravanti più forti in FIFA.png",
                "Ali sinistre più forti in FIFA.png",
                "Ali destre più forti in FIFA.png",
                "Portieri più forti in FIFA.png",
                "Analizza tutto"]
```

Come si vede, prendiamo in esame diverse caratteristiche che contraddistinguono i calciatori (la divisione e analisi viene svolta per nazione, età, per i più giovani, meno giovani, portieri/difensori centrali/centravanti/attaccanti/ali sinistre/ali destre più forti) o per squadre (più popolari, mediamente più giovani e meno giovani).

La ricerca scelta andrà a modificare anche i log inseriti nel file di log creato prima.

```
for i in range(1,len(sys.argv)):
    argomento = sys.argv[i]
    if argomento == "-opzioni":
        for j, opzione in enumerate(opzioni):
            print("{}\n {}".format(opzione,descr_opzioni[j]))
        quit()
    if argomento == "-gn":
        logger.info("chiamata funzione giocatori_nazione")
        giocatori_nazione()
    if argomento == "-ng":
        logger.info("chiamata funzione numero_giocatori")
        numero_giocatori()
    if argomento == "-sp":
        logger.info("chiamata funzione squadre_popolari")
        squadre_popolari()
    if argomento == "-gg":
        logger.info("chiamata funzione giocatori_giovani")
        giocatori_giovani()
    if argomento == "-gv":
        logger.info("chiamata funzione giocatori_vecchi")
        giocatori_vecchi()
```

L'output dell'analisi permette di ottenere, nella cartella vista prima, il salvataggio di file .png o file .csv con l'esito della ricerca effettuata sul dataset di Fifa.

Sotto riportiamo alcuni esempi tratti dalle 14 analisi disponibili.

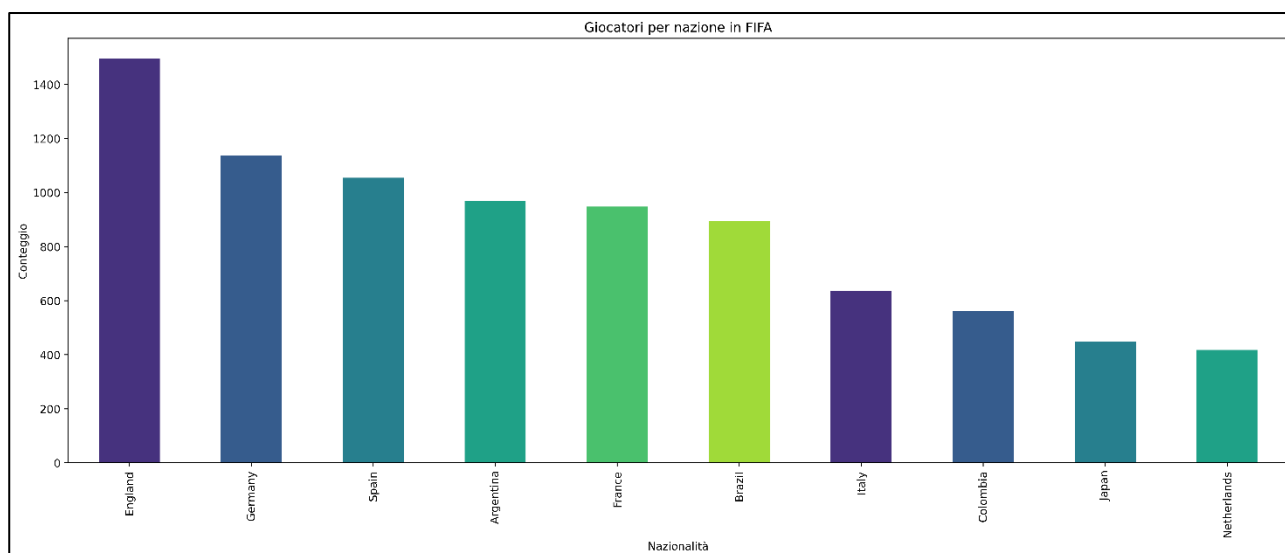


Figura 3 – Giocatori per nazioni

Nella Figura 3 si nota che i giocatori per nazione in Fifa più numerosi sono rispettivamente inglesi e tedeschi. La prima nazione non europea è l'Argentina, al terzo posto, mentre l'Italia in questa classifica è al settimo.

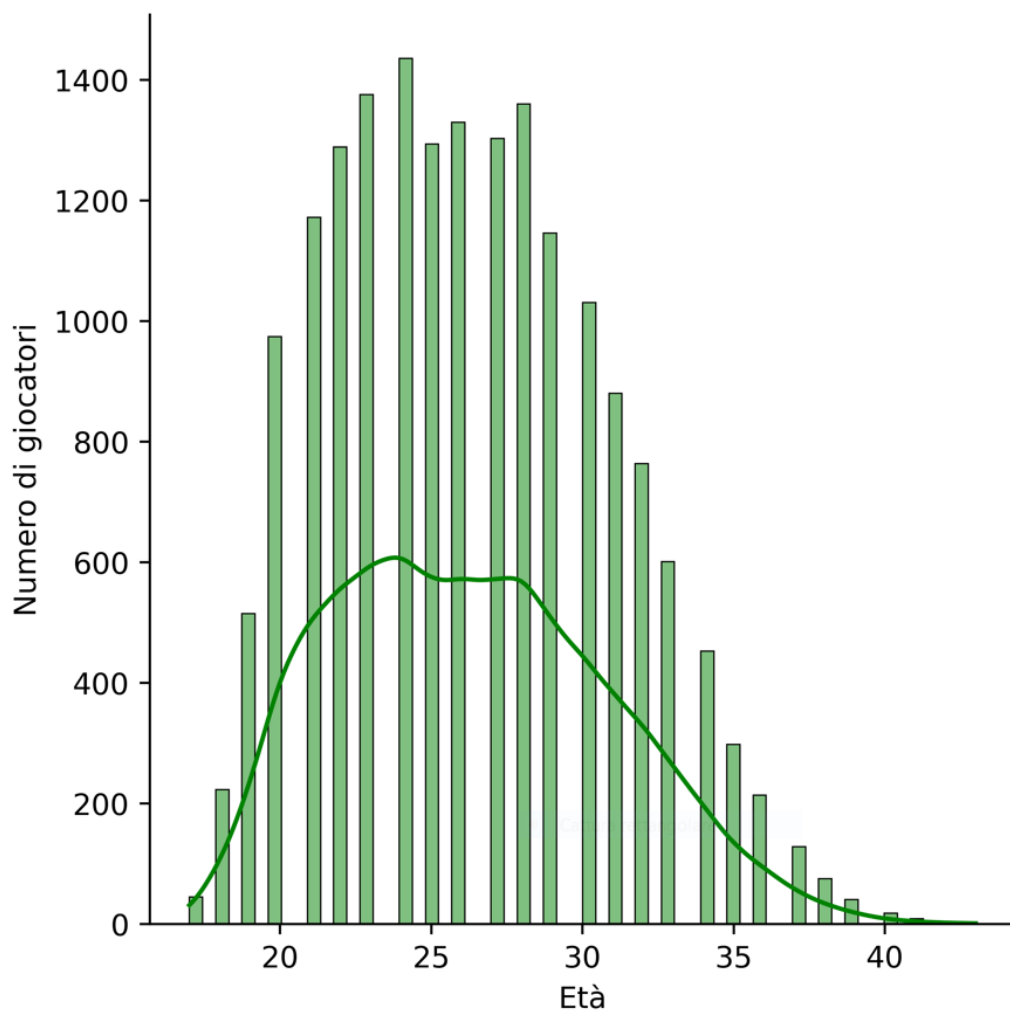


Figura 4 – Giocatori per età

La Figura 4 è indicativa dell'età dei giocatori. Come ci si aspettava il terzo decennio è quello con maggiore distribuzione, ma anche ben oltre dopo i 30 anni si può essere competitivi e, in alcuni casi, anche intorno ai 40. Gli outlier si pongono rispettivamente a 17 e 43 anni. A questo punto vogliamo vedere chi sono questi outlier riguardo ai meno giovani.

name	age	team	nationality
Hussain Omar Sulaimani	43	Al Ahli	Saudi Arabia
Leao Butran	43	Club Alianza Lima	Peru
Luis Fernando Fernandez	42	Sociedad Deportiva Aucas	Colombia
Gianluigi Buffon	42	Juventus	Italy
Hilton	42	Montpellier Herault SC	Brazil
Cristian Lucchetti	42	Atletico Tucumajn	Argentina
Jean-François Gillet	41	Standard de Liege	Belgium
Lee Dong Gook	41	Jeonbuk Hyundai Motors	Korea Republic
Michael Gurski	41	SpVgg Unterhaching	Germany
Robinson Zapata	41	Jaguars Futbol Club	Colombia

Figura 5 – Classifica dei giocatori meno giovani

Nella Figura 5 riportiamo i primi 10 giocatori per età, a partire dal meno giovane. Come si vede, tra questi ultraquarantenni vi è anche l'italiano Gianluigi Buffon.

name	age	team	nationality	overall
Lionel Messi	33	FC Barcelona	Argentina	94
Cristiano Ronaldo	35	Juventus	Portugal	93
Neymar Jr	28	Paris Saint-Germain	Brazil	92
Virgil van Dijk	29	Liverpool	Netherlands	91
Jan Oblak	27	Atletico Madrid	Slovenia	91
Kevin De Bruyne	29	Manchester City	Belgium	91
Robert Lewandow	31	FC Bayern Munchen	Poland	91
Eden Hazard	29	Real Madrid	Belgium	91
Alisson	27	Liverpool	Brazil	90
Mohamed Salah	28	Liverpool	Egypt	90

Figura 6 – Giocatori più forti

A questo punto siano attratti dalla classifica dei più forti. Nella Figura 6 vediamo in Excel la classifica dei giocatori più forti in base al punteggio "overall". Al primo posto abbiamo Lionel Messi con uno score di 94 seguito da Cristiano Ronaldo. Nella classifica dei dieci giocatori più forti, ben 3 militano nel Liverpool. Degna di nota è anche la presenza in questa top 10 di un giocatore africano.

Abbiamo qui sopra illustrato alcune delle analisi effettuate su nostro dataset e le abbiamo rievocate a titolo esemplificativo. In realtà, come accennato, sono state operate altre analisi (14 in tutto) per

identificare e comprendere le distribuzioni relative ai giocatori suddivisi per ruolo, per età, per nazione, come anche analisi relative alle squadre calcistiche, non solo in base a caratteristiche della loro composizione, ma anche come realtà inserite in un contesto sociale (es. squadre più o meno popolari).