UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ACCADEMIC YEAR 2022-2023 - SECOND SEMESTER

COURSE PAPER

Information Retrieval Systems

Player Scouting Recommendation System Using Similarity Methods and Generative AI

Professor:

Prof. Ing. Antonio Maria Rinaldi

Student:

Antonio Romano M63001315

Contents

1 Introduction **1**

1.1 Data Source . . . 1

1.2 Technologies utilized . . . 2

1.3 Methodological workflow of the overall system . . . 3

2 Data preprocessing **4**

2.1 Data Retrieval . . . 5

2.2 Data Cleaning . . . 5

2.3 Feature Selection . . . 5

2.4 Data Aggregation and Transformation . . . 5

2.5 Data Export on Apache Solr . . . 5

2.5.1 Apache Solr and custom configuration for search . . . 6

2.5.2 Simple Search Engine with Solr and Streamlit . . . 8

3 Similar Players Component **10**

3.1 General methodological workflow of Similar Player Component . . . 10

3.2 Cosine Similarity . . . 11

3.3 Kmeans Clustering . . . 14

3.3.1 Selection of the best K-value: Elbow Method . . . 14

3.3.2 Validating clustering techniques: Silhouette Score . . . 17

3.4 Evaluation of the methods used . . . 19

3.4.1 Human Evaluation . . . 19

3.4.2 Rank Correlation Evaluation . . . 19

4 Scouter AI component **22**

4.1 Generative AI . . . 22

4.2 Methodological workflow of Scouter AI component . . . 23

4.3 Prompting Techniques . . . 24

4.3.1 Zero Shot Prompting . . . 25

4.3.2 One Shot Prompt . . . 26

4.3.3 Augmented Generation One Shot Prompt . . . 26

4.3.4 Augmented Generated One Shot Prompt with System Role . . . 26

4.3.5 Implementation . . . 27

4.4 Evaluation . . . 29

4.4.1 Human Evaluation . . . 29

4.4.2 Rouge Evaluation . . . 31

4.4.2.1 ROUGE-1 . . . 31

4.4.2.2 ROUGE-2 . . . 31

4.4.2.3 ROUGE-L . . . 32

4.4.2.4 Computing Python . . . 32

4.4.3 BERTScore Evaluation . . . 32

4.4.4 Results . . . 34

4.5 Overall methodological architecture of the Player Scouting Recommendation System . . . 36

5 Conclusion **37**
5.1 Application . . . 37
5.2 Pro and Cons . . . 38
5.3 Limits . . . 39
5.4 Future Goals . . . 39

List of Figures **41**

List of Tables **42**

Bibliography **43**

1 Introduction

Chapter Contents

1.1 Data Source 1
1.2 Technologies utilized 2
1.3 Methodological workflow of the overall system 3

The following documentation is aimed at developing a system for recommending professional football players using methodologies and technologies in the field of information retrieval. Specifically, the objective is to create a system that, starting from a player that a team is looking for, calculates the **10 players most similar to the indicated player**. From these, the system, using generative AI, recommends the player best suited to the team's characteristics by **generating a report**.

1.1 Data Source

The data used for this project was collected by **FBRef**[1] [1], a website that provides football statistics and information. FBRef has a database of over 200,000 players and teams and tracks a range of statistics. In particular, for the purposes of the paper, the author chose to collect statistics from this site on all players who played in the five major European leagues in the 2022-2023 season. The statistics include goals scored, assists, passes completed, tackles won and a number of other performance measures important for the design of the recommendation system. Here are some features taken:

- Rk: The player's id
- Player: The player's name
- Nation: The player's national team
- Pos: The player's position
- Squad: The player's team
- Comp: The league in which the player plays
- Age: The player's age
- Born: The player's date of birth
- MP: The minutes played by the player
- Starts: The minutes played by the player as a starter
- Min: The total minutes played by the player (including substitutes)
- SCA: The total number of "Shot-Creating Actions" performed by the player. It is a statistical metric used in football to measure a player's involvement in actions that directly lead to a shot attempt by their team.
- GCA: The total number of "Goal-Creating Actions" performed by the player. It is a statistical metric in football that measures a player's involvement in actions that directly lead to a goal scored by their team.

[1] FBRef, provided by Opta Sports, employs a comprehensive approach to gather player data. This includes manual data collection by trained analysts who observe matches, computer vision technology tracking player movements via cameras and sensors, resulting in performance metrics. Advanced algorithms yield insights like expected goals (xG).

- Total TotDist: The total distance covered by completed passes made by the player. This metric represents the total distance in meters or yards covered by the ball through all completed passes made by the player during the match.
- Total PrgDist: The total distance covered by completed passes that advance towards the opponent's goal. This metric represents the total distance in meters or yards covered by the ball through completed passes that advance into the opponent's area, contributing to the attack progression.
- KP: The number of "Key Passes" made by the player.
- 1/3: The number of passes completed in the final third of the opponent's field.
- PPA: The number of passes completed in the opponent's penalty area.
- CrsPA: The number of crosses completed in the opponent's penalty area.
- ... and other features

The author took approximately 121 features with 2889 Players.

1.2 Technologies utilized

This project was completely developed in **Python** (vers 3.10.2), employing several key libraries, each with a specific function:

- **LangChain** (vers. 0.0.275): This library was used to interact with OpenAI and manage the generative AI model, GPT-3.5. It enabled the system to generate detailed reports based on specific inputs.
- **Scikit Learn**: This was used to calculate the Cosine Similarity between players, as well as to run the K-Means algorithm, which helped divide players into clusters based on their characteristics.
- **Pandas** (vers. 1.4.3): This library was used for data manipulation, including the organisation and analysis of football player data.
- **Scipy** (vers. 1.9.3): This was used to calculate the correlation between players' rankings, allowing a more in-depth evaluation of their performance.
- **RougeScore** (vers. 0.1.2): This was used to assess the quality of the reports generated based on metrics comparing them with reference texts.
- **BERTScore** (vers. 0.3.13): This metric was used to calculate the semantic similarity between the text generated by the AI model and the reference text, providing an indication of the quality of generation.
- **Streamlit** (vers. 1.23.1): The Streamlit framework was adapted to the needs of the project to create an intuitive user interface. It was used to implement a player search engine with automatic suggestion functionality.

In addition, a simple player search engine was designed and implemented using **Apache Solr** (vers. 8.11.2), with interaction in Python managed through the **SolrClient** (vers. 0.3.1) library. Solr was used for efficient indexing and searching of player data, thus contributing to the overall user experience in the system.

1.3 Methodological workflow of the overall system

The paper outlines a workflow involving several key components. It begins with the collection of data from FBREF, as outlined in the previous section, followed by pre-processing of the data. The processed data is then exported to Apache Solr, using it appropriately for player research and player statistics. The **'Similar Players' component** employs vector models and clustering algorithms to analyse the similarities between players, with meticulous evaluation of the results. Simultaneously, the **'Scouter AI' component** uses GPT-3.5 to generate recommendation reports for ideal player selections. All components are integrated into a Python application using Streamlit, providing a user-friendly interface. This comprehensive approach enables efficient analysis and recommendation of players, improving decision-making processes in football scouting.

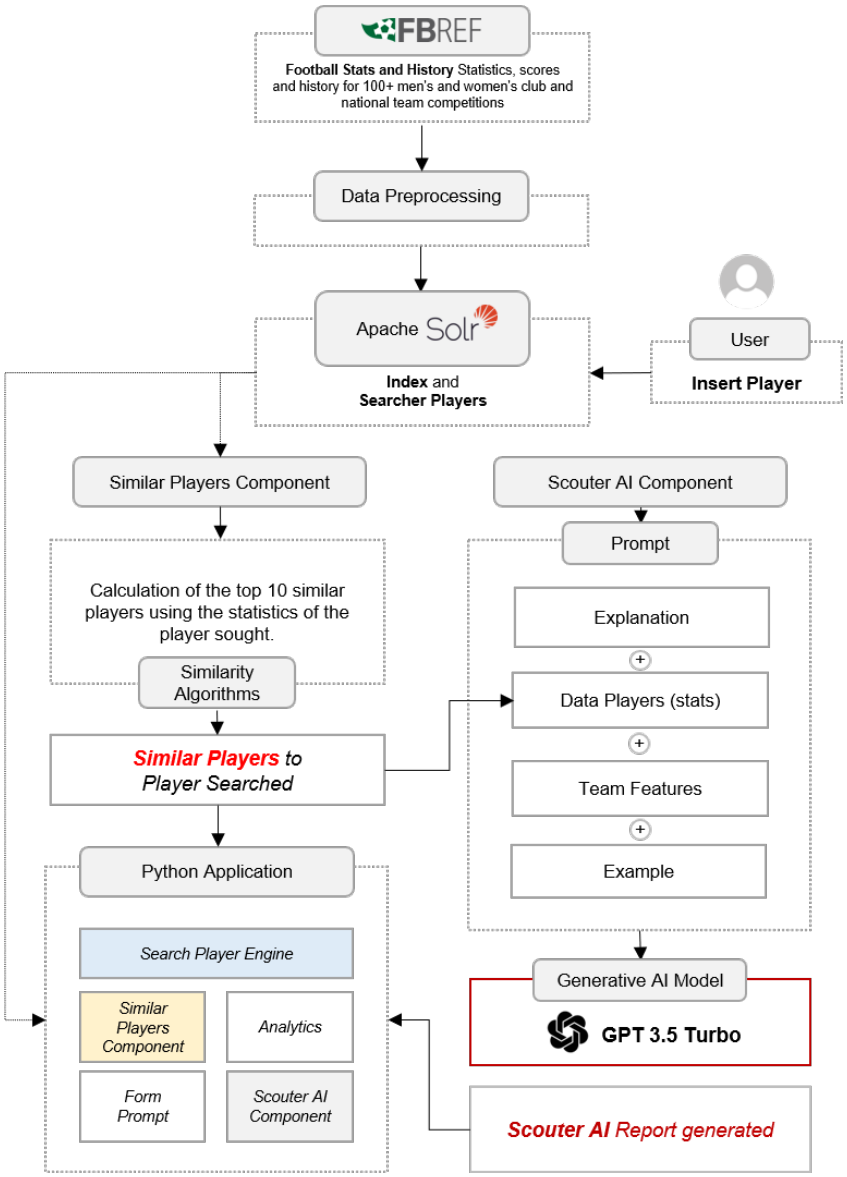

Figure 1.1: Methodological Workflow - Player Scouting Recommendation System

2 Data preprocessing

Chapter Contents

2.1 Data Retrieval 5
2.2 Data Cleaning 5
2.3 Feature Selection 5
2.4 Data Aggregation and Transformation 5
2.5 Data Export on Apache Solr 5
2.5.1 Apache Solr and custom configuration for search 6
2.5.2 Simple Search Engine with Solr and Streamlit 8

Prior to the start of the project, it is essential to undertake a data pre-processing phase in order to improve the quality of the data to be used in the subsequent development phases of the player Scouting recommendation system. During this phase, a series of data cleaning and preparation tasks are performed to ensure that the data are consistent and ready for further processing. This includes identifying and correcting any errors, handling missing data, removing redundant information and standardising data formats. The ultimate goal is to obtain a consistent and reliable set of data that can be used effectively in the subsequent phases of the project.

Below is the workflow followed for data preprocessing [1]:

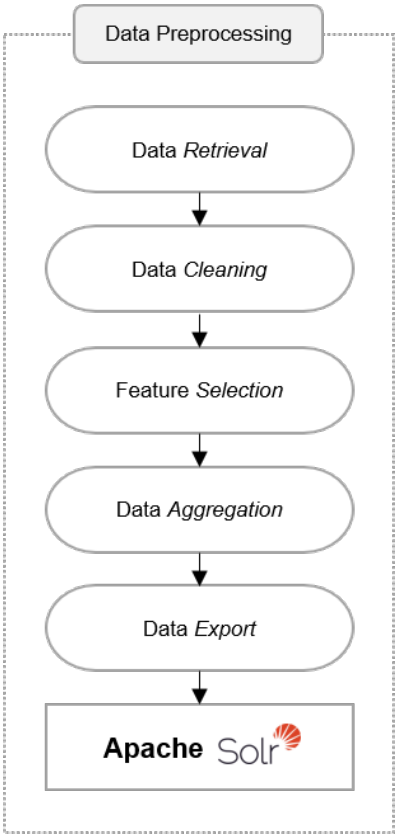

Figure 2.1: Workflow - Data Preprocessing

[1] The entire code is present in the *attached* **Data_Preprocessing.ipynb** notebook

2.1 Data Retrieval

The initial phase involved retrieving data from the Football Reference portal (FBRef). Data were collected using the 'Extract from web site' functionality of Excel. The data represent the statistics of a specific football player according to his position (e.g. goalkeeper, striker). This data was then imported into pandas DataFrames for further processing. To ensure data consistency, all rows containing irrelevant data were removed from consideration. This initial step ensured the acquisition of structured raw data.

2.2 Data Cleaning

In the data cleaning phase, rigorous quality control measures were applied. Duplicate rows were identified and removed to prevent redundancy, ensuring the dataset remained concise. Additionally, any missing or incomplete data entries, often denoted as NaN (Not a Number), were addressed using appropriate techniques. To maintain uniformity and facilitate subsequent computations, data type conversions were performed, ensuring all values adhered to a consistent numeric format.

2.3 Feature Selection

After cleaning the data, features were selected in order to identify and retain only the most relevant attributes. This was achieved by calculating and comparing the variances between the features in the dataset. Features with a variance below a predefined threshold were omitted, resulting in a more focused and efficient dataset. In addition, other columns that in context may be unnecessary for the construction of the recommendation system are eliminated.

2.4 Data Aggregation and Transformation

The next step involves the aggregation of data according to individual players. Using aggregation functions such as sum and max, player-specific statistics were consolidated into complete player profiles without hypothetical repetition as players were duplicated due to the hypothetical change of team in the same year. The dataset was then meticulously sorted, ensuring an orderly representation to facilitate analysis. The "Nation" and "Competion" columns were appropriately 'transformed' to enhanced the dataset's uniformity and prepared it for further analysis.

2.5 Data Export on Apache Solr

After data processing, the final data set was imported into **Apache Solr**. Solr is commonly used for indexing and searching large datasets, which makes it a valuable tool for data retrieval and analysis. However, Apache Solr has no official Python documentation and libraries, particularly for version 3.10. The choice was made to use the Python SolrClient library for data import and retrieval. Here is the interaction between App-SolrClient-Apache Solr:

Figure 2.2: ClienAPP, SolrClient, Apache Solr

Code extract showing Python - Apache Solr interaction with SolrClient:

```python
# library used
from SolrClient import SolrClient
import json

# data preprocessed
docs = dataframe_preprocessed.to_dict(orient='records')

# Creating a Solr client object that connects to the Solr server running on
    localhost and port 8983.
solr = SolrClient('http://localhost:8983/solr')

# Indexing JSON data into the "FootballStatsCore" core on Solr.
# The variable "docs" contains the data to be indexed, converted to JSON
    format.
solr.index_json("FootballStatsCore", json.dumps(docs))

# Performing a commit to make the indexing changes permanent.
solr.commit("FootballStatsCore", softCommit=True)
```

Figure 2.3: Data Export on Apache Solr with SolrClient Library Python

Note: Before performing this operation, Apache Solr must be appropriately configured.

2.5.1 Apache Solr and custom configuration for search

Apache Solr is based on Apache Lucene and is entirely written in Java. It exposes Lucene's functionalities through REST requests, utilizing them for document indexing and search implementation. An important aspect of Solr is its "schemaless" nature, allowing flexible schema definition for documents using XML after the creation of the Solr Core. The Core is a logical unit representing a single index and its configuration files like *solrconfig.xml* and *managed-schema*.

In this case, a Solr core named 'FootballStatsCore' was created. The configuration of the schema was performed using the Solr ADMIN UI (Dashboard Localhost:8983). The search engine was customised by adding query analysis and indexing features.

The `'solr.ASCIIFoldingFilterFactory'` [2] filter was added to allow searches without accents, improving the user experience when retrieving data, as some players have accents and special characters that can interfere with the search.

E.G (Player Search): Rafael Leao $\xrightarrow{\texttt{solr.ASCIIFoldingFilterFactory}}$ Rafael Leão

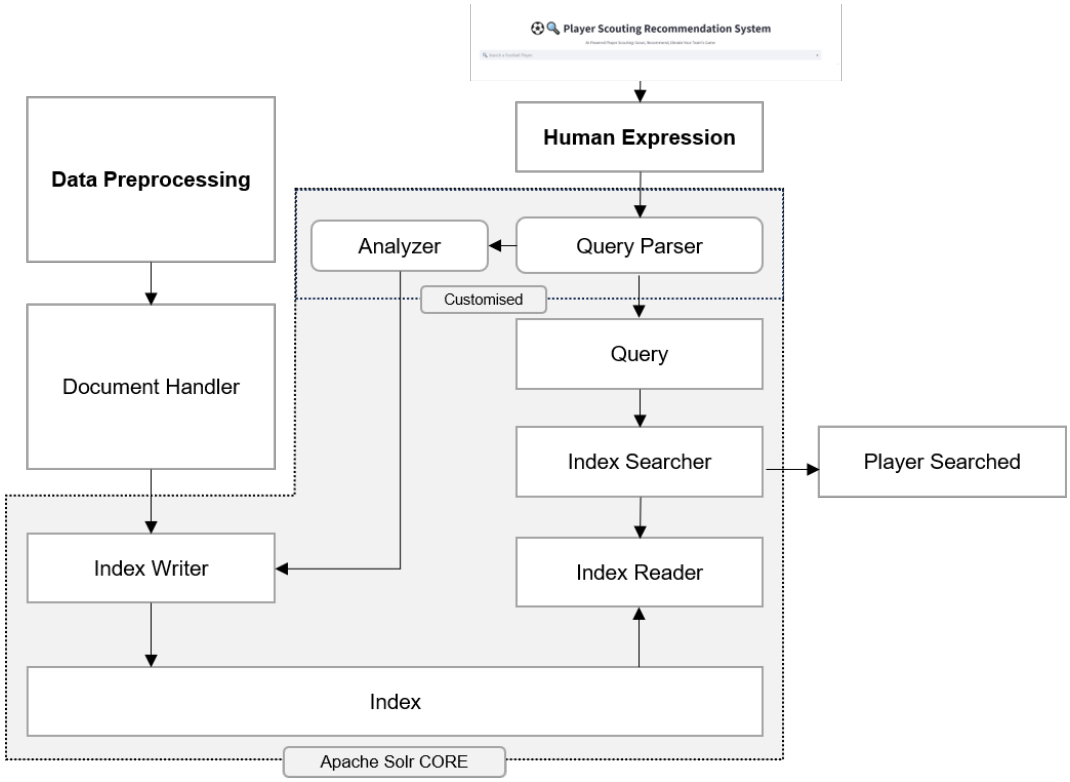

Figure 2.4: Apache Solr Architecture Scheme for FootballStatsCore

In the Figure 2.4, the components that make up the architecture are:

- **Document Handler**: Manages the extraction of data from the data preprocessing phase.
- **Index Writer**: Analyzes and stores documents in the index. It uses the Analyzer as a strategy to improve the indexing process.
- **Analyzer**: Filters the text coming from the query parser. In the specific case, it applies a filter for accented characters.
- **Query Parser**: Converts the query text into a Query object, which represents the search request. This process involves syntactic and semantic analysis of the query text to identify search criteria.
- **Index Searcher**: Forms the core of the search process. It uses the IndexReader to access the index and retrieve all terms matching the Query object. Finally, it returns the search results.

Here is an extract of the XML schema configuration in Solr:

```
<schema name="default-config" version="1.6">
  <!-- Field Type Definitions -->
  <fieldType name="text_general" class="solr.TextField"
      positionIncrementGap="100">
    <analyzer type="index">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.ASCIIFoldingFilterFactory"/> #added
    </analyzer>
    <analyzer type="query">
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.ASCIIFoldingFilterFactory"/> #added
    </analyzer>
  </fieldType>

  <!-- Field Definitions -->
  <field name="Rk" type="int" indexed="true" stored="true"/>
  <field name="Player" type="text_general" indexed="true" stored="true"/>
  <!-- Added other fields here... -->
</schema>
```

Figure 2.5: Code extract 'Managed Schema' file for the configuration of the Apache Solr Core - FootballStatsCore

2.5.2 Simple Search Engine with Solr and Streamlit

The first component of the system is the simple search engine for finding the player and his characteristics. The idea is to implement a suggestion query system with auto-completion. A streamlit component (st_searchbox)[2] and the SolrClient library were used for this purpose.
Below is Figure 2.6 demonstrating how it works:

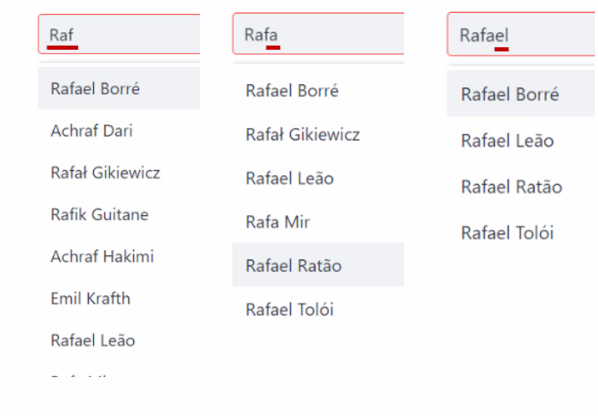

Figure 2.6: Autocomplete system implemented with streamlit component and SolrClient

[2] From https://github.com/m-wrzr/streamlit-searchbox

The `search_solr` function performs a search in the Solr core (FootballStatsCore) using a search term entered by the user. The results are extracted and returned as player names. The realised search box allows users to enter one letter at a time, showing suggestions and results as they come up. This integration simplifies the search for football players within the system, utilising Solr as the search engine and Streamlit for the user interface, improving the overall user experience when searching for desired players.

Below is the extract code of the implemented autocomplete search engine:

```
import streamlit as st
from streamlit_searchbox import st_searchbox
from SolrClient import SolrClient
from typing import List

# function that is called by the component whenever the user types a letter
    in the search box
def search_solr(searchterm: str) -> List[any]:
    if searchterm:
        res = solr.query('FootballStatsCore', {
            'q': 'Player:' + '*' + searchterm + '*',
            'fl': 'Rk,Player',
            'rows': 100000,
        })
        result = res.docs

        if(result != []):
            df_p = pd.DataFrame(result)
            df_p['Rk'] = df_p['Rk'].apply(lambda x: x[0])
            df_p['Player'] = df_p['Player'].apply(lambda x: x[0])
            return df_p['Player']
        else:
            return []

# Streamlit search box
selected_value = st_searchbox(
    search_solr,
    key="solr_searchbox",
    placeholder="Search a Football Player"
)
```

Figure 2.7: Solr with Streamlit autocompletation - Code Extract

3 Similar Players Component

Chapter Contents

3.1 General methodological workflow of Similar Player Component 10
3.2 Cosine Similarity 11
3.3 Kmeans Clustering 14
3.3.1 Selection of the best K-value: Elbow Method 14
3.3.2 Validating clustering techniques: Silhouette Score 17
3.4 Evaluation of the methods used 19
3.4.1 Human Evaluation 19
3.4.2 Rank Correlation Evaluation 19

With the data pre-processed as described in the previous section, we proceeded to the design phase of the recommendation system. To determine the most suitable method for comparing football players and assessing their similarity, two distinct approaches were considered: **cosine similarity** and **clustering with cosine similarity**.

3.1 General methodological workflow of Similar Player Component

The workflow highlights the steps involved in developing the ”Similar Players” component. Notably, it emphasizes the evaluation of algorithms to determine the most suitable one. If the evaluation results do not meet expectations, adjustments are made to the feature selection during the data preparation phase of the component. This iterative process ensures that the chosen algorithm aligns with the desired outcomes, optimizing the ”Similar Players” functionality for player analysis.

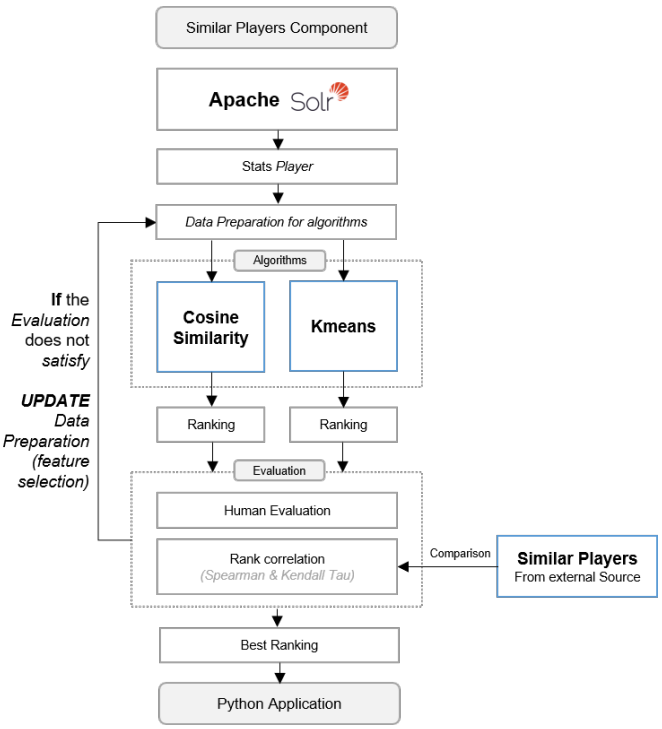

Figure 3.1: Workflow - Similar Players Component

3.2 Cosine Similarity

Cosine similarity [3] is a mathematical measure used to quantify the similarity between two non-zero vectors in a multi-dimensional space, particularly in the context of information retrieval and recommendation systems. It assesses how closely the direction of two vectors align in this high-dimensional space.
In many applications, particularly in recommender systems, data entities (such as documents, users or objects) are represented as vectors in a multidimensional space. Each dimension of this space corresponds to a specific characteristic or attribute of the entities. In the present case, in a football player recommendation system, each player is represented as a vector in which the dimensions represent various player statistics (e.g. goals scored, assists, age, playing time, etc.).

The formula used by Cosine Similarity is as follows:

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{3.1}$$

Where:

- **A** and **B** are vectors representing entities in a multi-dimensional space. In the context of recommendation systems, these could be user or item vectors.
- $\cdot$ denotes the dot product of the vectors **A** and **B**.
- $\|\mathbf{A}\|$ represents the magnitude of vector **A**.
- $\|\mathbf{B}\|$ represents the magnitude of vector **B**.

In Figure 3.2 2D representation of cosine similarity:

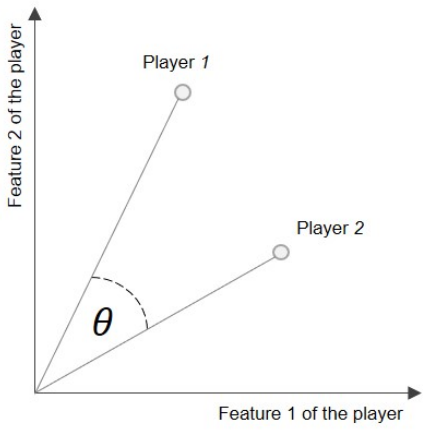

Figure 3.2: Cosine Similarity - Mathematical representation

When working with datasets consisting solely of vectors containing positive components, the cosine similarity metric typically returns values within the range of 0 to 1. Specifically:

- A cosine similarity of 1 indicates that the vectors are perfectly aligned, representing a state of complete similarity.
- A cosine similarity of 0 signifies orthogonality between vectors, suggesting a complete lack of similarity.

- Values between 0 and 1 indicate partial similarity between vectors, with higher values indicating a stronger degree of resemblance.

To improve results when calculating cosine similarity, *normalisation* of the data is preferred. This is because cosine similarity measures the angle between the data vectors, and normalisation ensures that features have comparable values. Normalisation is particularly useful when features have different scales or very large variations.
In this case, the MinMaxScaler Normalisation was used.

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3.2}$$

Where:

- X_{norm} is the normalised value of the feature.
- X is the original value of the feature.
- X_{min} is the minimum value of the feature in the entire data set.
- X_{max} is the maximum value of the feature in the entire data set.

This formula is used to normalize a feature so that the normalized values fall between 0 and 1 while preserving the relative proportions between the original values of the feature.
The code extract [1] used to apply cosine similarity to the data is described below:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity

# Initializing the scaler for data normalization
scaler = MinMaxScaler()
# Normalizing player data
df_player_norm[selected_features] =
    scaler.fit_transform(df_player_norm[selected_features])
# Calculating cosine similarity among the normalized players
similarity = cosine_similarity(df_player_norm[selected_features])
# Retrieve the index (Rk) of the chosen player ('choice') from the DataFrame
    'df_player'
index_player = df_player.loc[df_player['Player'] == choice, 'Rk'].values[0]
# Calculate the similarity scores between the chosen player and all other
    players
similarity_score = list(enumerate(similarity[index_player]))

# Sort the players based on their similarity scores in descending order
similar_players = sorted(similarity_score, key=lambda x: x[1], reverse=True)
```

Figure 3.3: Cosine Similarity for Similar Players - Code Extract

This results in the cosine similarity matrix in the Figure 3.4 and the evaluated ranking in the Figure 3.1:

[1] The entire code is present in the *attached* **SimilarPlayers_ScouterAI_Design_Evaluation.ipynb** notebook

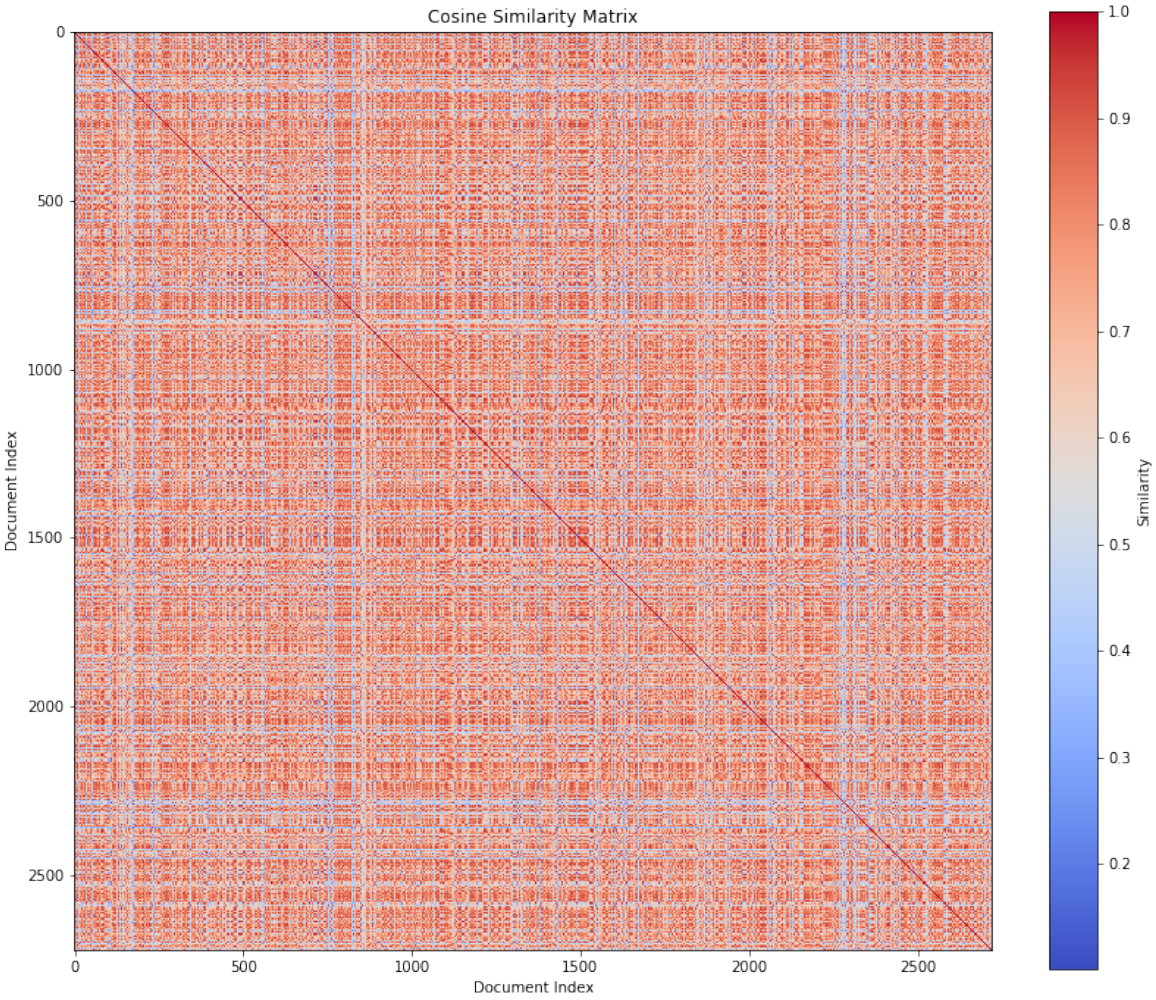

Figure 3.4: Cosine Similarity Matrix

Rk	Player	Nation	Pos	Squad	Comp	Age
1892	Victor Osimhen	NGA	FW	Napoli	Serie A	25
1882	Loïs Openda	BEL	FW	Lens	Ligue 1	23
2644	Callum Wilson	ENG	FW	Newcastle Utd	Premier League	31
2608	Elye Wahi	FRA	FW	Montpellier	Ligue 1	20
1596	Lautaro Martínez	ARG	FW	Inter	Serie A	26
1278	Harry Kane	ENG	FW	Tottenham	Premier League	30
269	Wissam Ben Yedder	FRA	FW	Monaco	Ligue 1	33
1446	Robert Lewandowski	POL	FW	Barcelona	La Liga	35
1828	Christopher Nkunku	FRA	FW,MF	RB Leipzig	Bundesliga	26
2085	Marcus Rashford	ENG	FW	Manchester Utd	Premier League	26

Table 3.1: Ranking evaluated by Cosine Similarity - Example on Erling Haaland

3.3 Kmeans Clustering

The goal of clustering is to determine the internal grouping in a set of unlabeled data. In K-means clustering, our algorithm divides the data into K clusters and each cluster is represented by it's centroid/center, that is the mean of the players in that cluster. In order to classify the players into groups, it is need a way to compute the (dis)similarity between each pair of players. The result of this computation is known as a dissimilarity or distance matrix. Also in this case, it is used the cosine similarity

- **Initialization**: Initially, you specify the number of clusters (K) it want to create. Then, K initial cluster centroids are randomly selected from the data points or set randomly within the feature space.
- **Assignment**: Each data point is assigned to the cluster whose centroid is closest in terms of some distance metric. This assignment is based on the similarity between data points and cluster centroids.
- **Update**: After all data points have been assigned to clusters, the centroids of the clusters are updated. The new centroids are calculated as the mean of all data points belonging to that cluster.
- **Iteration**: Steps 2 and 3 are repeated until a stopping criterion is met. Common stopping criteria include a maximum number of iterations, minimal change in cluster assignments, or minimal change in cluster centroids.
- **Result**: Once the algorithm *converges or the stopping criterion* is met, the final clusters are obtained, and each data point belongs to one of these clusters.

3.3.1 Selection of the best K-value: Elbow Method

The **Elbow Method** [4] is a graphical technique used to determine the optimal number of clusters (K) in a K-means clustering analysis. It relies on the sum of squared errors (SSE), also known as the "within-cluster variance" or "inertia," as a measure of clustering quality. The formula to calculate SSE for a specific K is as follows:

$$SSE(K) = \sum_{i=1}^{N} \sum_{j=1}^{K} ||x_i - \mu_j||^2 \tag{3.3}$$

Where:

- N is the total number of data points.
- K is the number of clusters.
- x_i represents a data point.
- μ_j represents the centroid of cluster j.
- x_i represents a data point.
- $||x_i - \mu_j||^2$ represents the squared Euclidean distance between x_i and μ_j.

The Elbow Method follows these steps:

1. **Initialization**: Start by selecting a range of potential values for K, typically from 1 to some upper limit, depending on the problem and dataset.

2. **K-means Clustering**: Apply the K-means clustering algorithm to the dataset for each value of K in the specified range.
3. **SSE Calculation**: For each K, compute the SSE using Equation (1).
4. **Elbow Plot**: Create a plot of SSE values for each K. The x-axis represents the number of clusters (K), and the y-axis represents the corresponding SSE values.
5. **Selecting** K: Analyze the SSE vs. K plot to identify the "elbow point" where the SSE starts to level off or decrease at a slower rate. This point indicates the optimal number of clusters for the dataset.
6. **Final Clustering**: Once the optimal K is determined, perform the final K-means clustering with that K value to group the data accordingly.

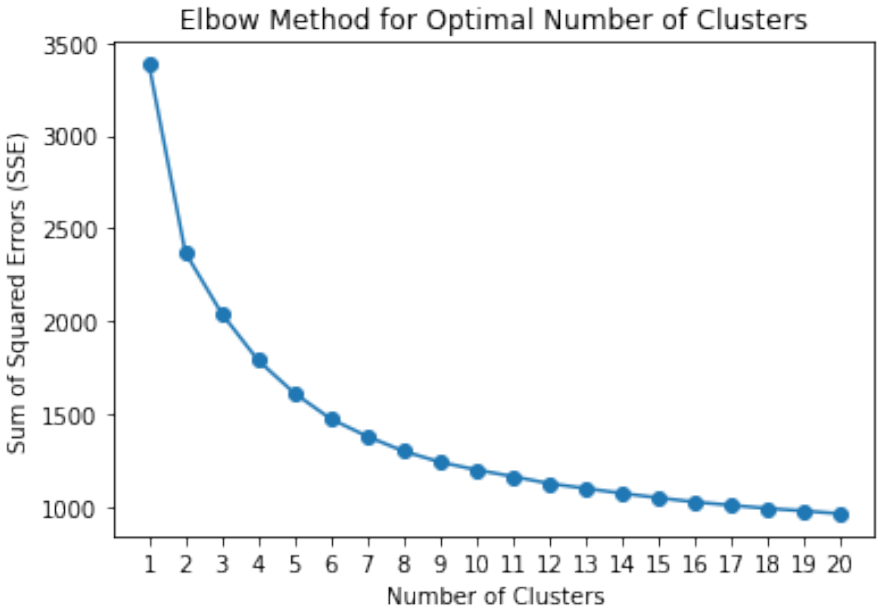

Figure 3.5: Elbow Method for optimal value of k in KMeans

In this case, **k = 4** (the knee of the curve) was chosen as the number of clusters.

From the extracted code[2], the sum of the quadratic errors (SSE) is calculated for a range of different cluster values from 1 to 20 to determine the optimal bending point of the Elbow Curve, which in this case is 4 as the desired number of clusters (Figure 3.5 shows). Next, the K-Means model is trained on the characteristics of the players, and each player is assigned a cluster label based on its similarity to the other players in the same cluster.

[2]The entire code is present in the *attached* **SimilarPlayers_ScouterAI_Design_Evaluation.ipynb** notebook

```
# Kmeans clustering
# Import necessary libraries
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

# Calculate SSE (inertia) for different cluster values
sse_values = []
possible_clusters = range(1, 21) # Try a range of cluster values from 1 to 20

for num_clusters in possible_clusters:
    # Create a K-Means model with the current number of clusters
    kmeans_model = KMeans(n_clusters=num_clusters, random_state=42)

    # Fit the K-Means model to the normalized player features
    kmeans_model.fit(df_player_norm[selected_features])

    # Append the Sum of Squared Errors (SSE) to the list
    sse_values.append(kmeans_model.inertia_)

# ... plotting Elbow method ...
plt.plot(possible_clusters, sse_values, marker='o')

# Specify the desired number of clusters on the basis of the elbow function
    knee
num_clusters = 4

# Create an instance of the K-Means model with the specified number of
    clusters
kmeans_model = KMeans(n_clusters=num_clusters, random_state=42)

# Train the K-Means model on the normalized player features
kmeans_model.fit(df_player_norm[selected_features])

# Get cluster labels assigned to players
cluster_labels = kmeans_model.labels_
```

Figure 3.6: Kmeans and Elbow method for selection K - Code Extract

3.3.2 Validating clustering techniques: Silhouette Score

To calculate the goodness of a clustering technique, the **Silhouette Coefficient** [5] or silhouette score was used. Its value ranges from -1 to 1.

- 1: Means clusters are well apart from each other and clearly distinguished.
- 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.
- -1: Means clusters are assigned in the wrong way.

The formula for the Silhouette Score for an individual cluster *i* where i ranges from 1 to n and n is the number of clusters:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{3.4}$$

Where:

- S_i: is the Silhouette Score.
- a_i: average intra-cluster distance i.e the average distance between each point within a cluster.
- b_i: average inter-cluster distance i.e the average distance between all clusters.
- The denominator $\max(a_i, b_i)$ is used to ensure that S_i always falls within the range [-1, 1].

Results of Silhouette Score:

N_Cluster	Silhouette Score
3	0.256
4	0.274
5	0.242
6	0.190

Table 3.2: Silhouette Score

Among the tested numbers of clusters, the highest Silhouette Score is achieved when using 4 clusters.

A Silhouette Score of 0.274 suggests that the clustering with 4 clusters is relatively better in terms of both cohesion within clusters and separation between clusters compared to other numbers of clusters.

Given that the results are close to zero, it indicates that the distance between clusters is not significant. This could suggest that the clustering solution is not optimal. In other words, the results obtained with this clustering solution may not significantly improve compared to a situation where clustering was not applied.

The code extract used to apply cosine similarity to the data is described below:

```
# Calculate and print the Silhouette Score for a specified number of
    clusters (n=4)
print(f'Silhouette Score (n=4):
    {silhouette_score(df_player_norm[selected_features], cluster_labels)}')

# Select the player of interest (target)
target_player = df_player_norm[df_player_norm['Rk'] == index_player]
# Extract the target player's features
target_features = target_player[selected_features]

# Calculate the cosine similarity between the target player and other
    players in the same cluster
similarities = cosine_similarity(target_features,
    similar_players_cluster_df[selected_features])
similarities = similarities[0] * 100 # Scaling the similarity values from -1
    to 1 to a range of -100 to 100

# Sort the DataFrame based on the 'Similarity' column in descending order
similar_players_cluster_df =
    similar_players_cluster_df.sort_values(by='Similarity', ascending=False)
```

Figure 3.7: Calculation of Silhouette Score and Ranking - Code Extract

From the extracted code, the Silhouette Score is calculated and printed for a specified number of clusters (n=4) to validate the clustering technique. Next, the player of interest (target) is selected from the DataFrame and its characteristics are extracted. The cosine similarity between the player of interest and the other players in the same cluster is then calculated. Finally, the DataFrame of similar players in the cluster is sorted according to the 'Similarity' column in descending order.

The ranking obtained by the Kmeans is:

Rk	Player	Nation	Pos	Squad	Comp	Age
269	Wissam Ben Yedder	FRA	FW	Monaco	Ligue 1	33
1278	Harry Kane	ENG	FW	Tottenham	Premier League	30
1446	Robert Lewandowski	POL	FW	Barcelona	La Liga	35
1596	Lautaro Martínez	ARG	FW	Inter	Serie A	26
1828	Christopher Nkunku	FRA	FW,MF	RB Leipzig	Bundesliga	26
1882	Loïs Openda	BEL	FW	Lens	Ligue 1	23
1892	Victor Osimhen	NGA	FW	Napoli	Serie A	25
2085	Marcus Rashford	ENG	FW	Manchester Utd	Premier League	26
2608	Elye Wahi	FRA	FW	Montpellier	Ligue 1	20
2644	Callum Wilson	ENG	FW	Newcastle Utd	Premier League	31

Table 3.3: Ranking evaluated by Kmeans - Example on Erling Haaland

3.4 Evaluation of the methods used

To assess the results obtained from the two approaches, we employed Rank Correlation and Human Evaluation. Specifically, Rank Correlation was evaluated based on some ”Top-10 Similar Players” rankings provided by FBRef. Below are the two rankings obtained from the respective approaches:

Player	Nation	Squad
Wissam Ben Yedder	FRA	Monaco
Harry Kane	ENG	Tottenham
Robert Lewandowski	POL	Barcelona
Lautaro Martínez	ARG	Inter
Christopher Nkunku	FRA	RB Leipzig
Loïs Openda	BEL	Lens
Victor Osimhen	NGA	Napoli
Marcus Rashford	ENG	Manchester Utd
Elye Wahi	FRA	Montpellier

Table 3.4: Ranking from Kmeans

Player	Nation	Squad
Victor Osimhen	NGA	Napoli
Loïs Openda	BEL	Lens
Callum Wilson	ENG	Newcastle Utd
Elye Wahi	FRA	Montpellier
Lautaro Martínez	ARG	Inter
Harry Kane	ENG	Tottenham
Wissam Ben Yedder	FRA	Monaco
Robert Lewandowski	POL	Barcelona
Christopher Nkunku	FRA	RB Leipzig

Table 3.5: Ranking from Cosine Similarity

Table 3.6: Example of Erling Haaland

3.4.1 Human Evaluation

From the tables, differences between the rankings obtained from the two approaches are evident. The domain expert asserts that the ranking evaluated by Cosine Similarity has proven superior to that of Kmeans. The results appear to be more ”realistic” in the case of Cosine Similarity. Expanding the evaluations to include additional players tested further confirms the superiority of Cosine Similarity.

3.4.2 Rank Correlation Evaluation

We are now interested in assessing the disparities between the two approaches employed and a ranking derived from a reliable source like FBRef. To accomplish this, we utilized rank correlation metrics in particular ”The Spearman Correlation Coefficient”[6] and ”The Kendall Tau Correlation Coefficient”.
For two rankings, R_1 and R_2, the correlation value $C(R_1, R_2)$ falls within the range of -1 to 1.

- If $C(R_1, R_2) = 1$, the agreement between the two ranking is perfect (they are the same).
- If $C(R_1, R_2) = 0$, the two rankings are completely independent.
- If $C(R_1, R_2) = -1$, the disagreement between the two ranking is perfect (they are the reverse of each other).

Increasing values of $C(R_1, R_2)$ imply increasing agreement between the two rankings.

- **Spearman Coefficient**

$$\rho = 1 - \frac{6 \sum d^2}{n(n^2 - 1)} \tag{3.5}$$

  where:

  - $\sum d^2$ is the sum of the squared differences in ranks between corresponding elements in the two rankings.

  - n is the number of paired observations.

- **Kendall Tau Coefficient**

$$\tau = \frac{\text{Number of Concordant Pairs} - \text{Number of Discordant Pairs}}{\frac{1}{2}n(n-1)} \tag{3.6}$$

  where:

  - "Number of Concordant Pairs" counts the pairs of data points that maintain their relative order in both rankings.
  - "Number of Discordant Pairs" counts the pairs of data points that change their order between the rankings.
  - n is the number of paired observations.

```
from scipy.stats import spearmanr
from scipy.stats import kendalltau

spearmanr, _ = spearmanr(similar_players_cluster_df['Rk'].values,
    similar_reference_player_1)

kendall_tau, _ = kendalltau(similar_players_cluster_df['Rk'].values,
    similar_reference_player_1)
```

Figure 3.8: Spearman and Kendall Tau - Code Extract

The Spearman Coefficient is based on the differences between the positions of a same document in two rankings while Kendall's Tau Coefficient assesses how similar or in agreement two rankings are in the relative positions of the elements, without considering the exact differences between the positions.

Considering this rank provided by FBRef:

Player	Nation	Squad
Callum Wilson	ENG	Newcastle United
Dušan Vlahović	SRB	Juventus
Loïs Openda	BEL	RB Leipzig
Robert Lewandowski	POL	Barcelona
Cyle Larin	CAN	Mallorca
Terem Moffi	NGA	Nice
Folarin Balogun	USA	Monaco
Marcus Thuram	FRA	Internazionale
Victor Osimhen	NGA	Napoli
Ollie Watkins	ENG	Aston Villa

Table 3.7: Rank of top 10 similar players from **FBRef**

The results are:

- Cosine Similarity:
  - Spearman Coefficient: 0,454
  - Kendall Tau: 0,333
- Kmeans
  - Spearman Coefficient: -0,321
  - Kendall Tau: -0,288

Evaluating the rankings of two other players as well, the coefficients resulted:

Players	**Cosine Similarity**		**Kmeans**	
	Kendall Tau	Spearman	Kendall Tau	Spearman
Rafael Leao	0.022	0.03	0.288	0.333
Victor Osimhen	0.288	0.309	-0.022	0.042
Erling Haaland	0.333	0.454	-0.288	-0.321

Table 3.8: Other Results of Spearman and Kendall for each algorithm used

As can be seen from the results, cosine similarity is the best solution to construct the recommendation system as the values of the respective coefficients were better. In fact, there are more negative correlations in the Kmeans out of 3 evaluations than in the other solution. In addition, the use of Spearman is considered as it is necessary to evaluate the differences between the relative positions of the players and not only the exact positions, as is the case with Kendall Tau.

4 Scouter AI component

Chapter Contents

4.1 Generative AI . . . 22
4.2 Methodological workflow of Scouter AI component . . . 23
4.3 Prompting Techniques . . . 24
4.3.1 Zero Shot Prompting . . . 25
4.3.2 One Shot Prompt . . . 26
4.3.3 Augmented Generation One Shot Prompt . . . 26
4.3.4 Augmented Generated One Shot Prompt with System Role . . . 26
4.3.5 Implementation . . . 27
4.4 Evaluation . . . 29
4.4.1 Human Evaluation . . . 29
4.4.2 Rouge Evaluation . . . 31
4.4.3 BERTScore Evaluation . . . 32
4.4.4 Results . . . 34
4.5 Overall methodological architecture of the Player Scouting Recommendation System . . . 36

In this chapter, the implementation of the Scouter AI component is presented. In this phase, the GPT 3.5 generative model is exploited to create the recommended player report. This process automates the generation of detailed and informative reports based on player characteristics and team requirements.

4.1 Generative AI

Generative AI, also known as Generative Artificial Intelligence, represents a category of artificial intelligence designed for autonomously creating data or content based on existing high-quality data. In the specific case, we focus on Text Generation with AI[7], and one of the most renowned models in this field is the GPT (Generative Pre-Trained Transformer), developed by OpenAI. The model in question is GPT-3.5, which belongs to the GPT family.

GPT models, including Foundation Models, fall under the category of Large Language Models (LLMs)[8], known for leveraging deep learning in natural language processing (NLP) and natural language generation (NLG) tasks. To comprehend language complexity, these large-scale language models are pre-trained on extensive data. They address linguistic challenges through advanced techniques such as:

- Fine-tuning: Models are fine-tuned for specific tasks to enhance their performance in those tasks.
- Contextual Learning: They consider the context of previous words to determine the next output.
- Zero-shot/One-shot/Few-shot Learning: They can learn from a few examples or even no examples.

The primary goal of these models is to predict the next text in a sequence, achieved through probabilistic approaches. Each word in the sequence is generated based on the conditional probability of the previous words in the sequence.
The general formula for calculating the probability of generating a sequence of words (w1, w2, ..., wn) is as follows:

$$P(w1, w2, ..., wn) = P(w1) * P(w2|w1) * P(w3|w1, w2) * ... * P(wn|w1, w2, ..., wn - 1) \quad (4.1)$$

Where $P(w1)$ represents the probability of generating the first word w1, $P(w2|w1)$ the conditional probability of generating w2 given w1, and so on. This probabilistic model allows GPT-3.5 to generate word sequences.

To use language models efficiently, the model poses a hypothetical input to the user: the **prompt**.
A **prompt** [9] is a request or set of instructions given to the model to direct it to generate text. When the user enters a prompt, artificial intelligence applies statistical models to recognise the intention behind the request and to identify an outcome that is likely to be the desired one. One criticism levelled at these models is that they are designed to find the most average output, which means they are unable to create exceptional quality. Another criticism is that their results are only as good as the data they were trained on. If the training data were limited or contained biases or flaws, the results obtained will represent this.

In the present case, GPT-3.5 has been employed to generate a player Scouting recommendation report. This report is generated based on an understanding of the team's characteristics and identifying players who are similar to an ideal player.

The decision to use the GPT 3.5 model was based on a detailed analysis of the results obtained during comparative tests with other generative models. During these tests, it emerged that the GPT 3.5 Turbo model was able to generate reports in line with expectations, while ensuring an average execution time of only 24 seconds. This was very beneficial in terms of system efficiency and responsiveness.

Generative AI Models	Average Execution times (s)	Costs	Results
GPT 3.5 Turbo	24	$0.002 / 1K token	Report generated in line with expectations
LLAMA2/replicate	67	$0.000225 / sec	Report generated in line with expectations but sometimes exceeds token limit
LLAMA2/offline*	486	FREE	Good reports but too much waiting time
Cohere	75	$0.4 / 1M Tokens	Good reports but exceeds toker limit for each execution
Bard**	31	FREE	Reports not in line with expectations

Table 4.1: Choice of generative models

Furthermore, it should be noted that the cost associated with using GPT 3.5 Turbo is competitive, with a cost of $0.002 per 1,000 tokens generated. This was a deciding factor in the choice, as other models had significantly higher additional costs.

NOTE
- * *LLAMA2/offline* indicates that the model was run locally
- ** An unofficial API was used for the use of *Bard*

4.2 Methodological workflow of Scouter AI component

The steps in the following workflow illustrate the use of a prompt that will be presented to the GPT 3.5 model to generate a report. This report will then be evaluated through human evaluation or through automatic metrics such as Rouge Score and BERTScore. If the results obtained do not meet expectations, changes will have to be made to the prompt.

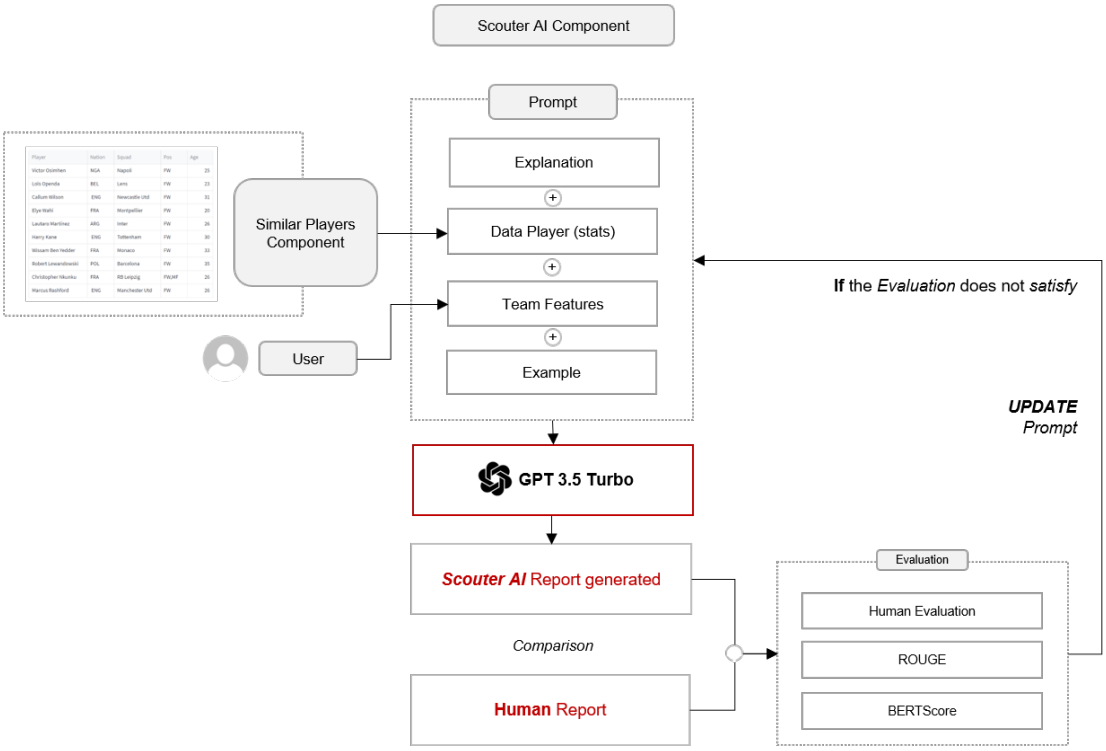

Figure 4.1: Methodological workflow of Scouter AI component

4.3 Prompting Techniques

To obtain better results from the generative model, various prompting techniques were used. In particular, a modular structure [10] was defined :

$$P = E + D + F + S \tag{4.2}$$

Where:

- **P**: **P**rompt;
- **E**: **E**xplanation: explanation of what is wanted from the model;
- **D**: **D**ata: data that the model has to process and summarise, in this case the top 10 similar players.
- **F**: **F**eature: team characteristics to be taken into account in the report to be generated.
- **S**: **S**how the type of the task: demonstration of an example of output useful for the model.

For each composition, a different technique was obtained such as:

- Zero Shot Prompting [11]
- One Shot Prompting [12]
- Augmented Generation One Shot Prompting [13]

the latter proposed by the author.

Below is the composition of the prompt used:

*Explanation (**E**)*

Generate a Football Talent Scout report based on the DATA PROVIDED (maximum 250 words) written in a formal tone FOLLOWING THE EXAMPLE.
It is essential to compare player attributes and select the most suitable candidate from the available options from among similar players, based on the TEAM REQUIREMENTS provided. It is important to note that the selection of players is not limited to the ranking of the players provided, as long as they meet the TEAM REQUIREMENTS.
THE PLAYER CHOSEN MUST NECESSARILY BE AMONG THE POSSIBLE PLAYERS CONSIDERED IN THE FOOTBALL SCOUT REPORT.
INDICATE the player chosen at the end of the REPORT.

*Data (**D**)*

DATA:

{content}
---

*Features (**F**)*

TEAM REQUIREMENTS:
Style of play: {style_t}
Player type required: {type_player}
Preferred league: {league}
Key ability: {ability}
Ideal formation: {formation}

*Show an Example (**S**)*

Report
After a detailed analysis of the data, we have identified candidates who best meet the requirements of your team. Below, we present three potential candidates:

Three potential candidates:
[Player X]: Highlights strengths and addresses weaknesses based on data on the essential attributes for a player in his specific age group.
[Player Y]: Highlights strengths and addresses weaknesses based on data regarding the attributes a player must necessarily possess in his specific age group.
[Player Z]: Highlighting strengths and addressing weaknesses based on attribute data that a player must necessarily possess in his specific age group.

[Provide the reasons for choosing the recommended player over the others].
The recommended player:
[INDICATE Name of player recommended]

Figure 4.2: Combination of Prompt

4.3.1 Zero Shot Prompting

The Zero Shot prompt is structured like this:

$$P = E + F \tag{4.3}$$

Where:

- **P**: **P**rompt;
- **E**: **E**xplanation: explanation of what you want to achieve from the model and names of similar top 10 players.
- **F**: **F**eature: team characteristics to be taken into account in the report to be generated.

It should be noted that in the above prompt, no specific examples are given to the model. This means that the model will generate the text based on its general knowledge without considering the characteristics of the players belonging to the top 10. Consequently, the structure of the report may deviate from expectations.

4.3.2 One Shot Prompt

The One Shot prompt is structured like this:

$$P = E + F + S \tag{4.4}$$

Where:

- **P**: **P**rompt;
- **E**: **E**xplanation: explanation of what you want to achieve from the model and names of similar top 10 players.
- **F**: **F**eature: team characteristics to be taken into account in the report to be generated.
- **S**: **S**how the type of the task: demonstration of an example of output useful for the model.

In this case, in addition to the input (E), an example of the report structure (S) is provided. The model will generate text following the structure provided in the example, but the content will be generated according to its knowledge. This approach results in a report with a defined structure, but with content generated autonomously by the model.

4.3.3 Augmented Generation One Shot Prompt

Augmented generation simply means adding external information to the input request entered into the LLM, thus augmenting the generated response. Therefore, this idea is exploited to add data from the Similar Player component so that the generated report will also consider the statistical data of the players resulting in the top 10. In this case, the structure of the prompt will be the complete combination:

$$P = E + D + F + S \tag{4.5}$$

Where:

- **P**: **P**rompt;
- **E**: **E**xplanation: explanation of what is wanted from the model;
- **D**: **D**ata: data that the model has to process and summarise, in this case the top 10 similar players.
- **F**: **F**eature: team characteristics to be taken into account in the report to be generated.
- **S**: **S**how the type of the task: demonstration of an example of output useful for the model.

The text that will be generated will be more accurate as it will also consider the players' statistics and justify the choices based on the data.

4.3.4 Augmented Generated One Shot Prompt with System Role

To guide the model more precisely on how it should respond, the role of the system is specified, allowing the model to better understand how it should respond to user requests.

- **System**: "You are a soccer scout and you must be good at finding the best talents in your team starting from the players rated by the similar player system."
- **User**:

$$P = E + D + F + S \tag{4.6}$$

  Where:

  - **P**: **P**rompt;
  - **E**: **E**xplanation: explanation of what is wanted from the model;
  - **D**: **D**ata: data that the model has to process and summarise, in this case the top 10 similar players.
  - **F**: **F**eature: team characteristics to be taken into account in the report to be generated.
  - **S**: **S**how the type of the task: demonstration of an example of output useful for the model.

An example of generation made from the following composition of prompts and GPT 3.5:

Generated Report

Football Talent Scout Report

After a thorough analysis of the data provided, we have identified several potential candidates who possess the necessary attributes to meet the requirements of your team. Based on the team requirements of adapting to ongoing game situations, solid skills, tactical knowledge, and a preference for tactical and defensive football, we have selected the following three potential candidates:

Three potential candidates:

Victor Osimhen: Victor Osimhen has shown solid skills and tactical knowledge acquired over time. He has a good aerial ability, speed, agility, and game intelligence. His finishing ability is also commendable, making him a threat in front of goal. Additionally, Osimhen has demonstrated leadership qualities on the field. He has the potential to adapt to different game situations and exploit opponent weaknesses.

Harry Kane: Harry Kane is an experienced player with a strong understanding of the game. He possesses excellent leadership skills, aerial ability, and game intelligence. Kane's finishing ability is exceptional, and he has the vision to create scoring opportunities for his teammates. His speed and agility allow him to be effective in different tactical setups. Kane's experience and tactical knowledge make him a valuable asset to any team.

Robert Lewandowski: Robert Lewandowski is a highly skilled and intelligent player. He has exceptional leadership qualities, aerial ability, and game intelligence. Lewandowski's finishing ability is world-class, and his vision on the field is outstanding. He possesses the speed and agility to adapt to different game situations. Lewandowski's experience and tactical knowledge make him a valuable asset to any team.

Based on the team requirements and the attributes of the potential candidates, the chosen player is Harry Kane. Kane's leadership skills, aerial ability, speed, agility, vision, finishing, and game intelligence make him the most suitable candidate for your team. He has the experience and tactical knowledge to adapt to different game situations and contribute significantly to the team's success.

Figure 4.3: Generated Report Example

4.3.5 Implementation

In the following code 4.4, we see that the process starts with the initialisation of the GPT-3.5 Turbo model, specifying some key settings, such as temperature. Temperature is a parameter that influences the creativity of the responses generated by the model. Setting the temperature to 0.3 indicates that the model is configured to produce relatively more conservative and consistent responses, suitable for a scenario where accuracy and consistency in responses is required.
Next, various prompt templates are defined, including a system message that presents the context to the model and a template for the prompt to be used. This prompt constitutes the input provided to the template, which will then be used to generate the desired report.

```
from langchain.chat_models import ChatOpenAI
from langchain.prompts.chat import (
    ChatPromptTemplate,
    SystemMessagePromptTemplate,
    AIMessagePromptTemplate,
    HumanMessagePromptTemplate,
)
from langchain.schema import AIMessage, HumanMessage, SystemMessage
chat = ChatOpenAI(temperature=0.3, model='gpt-3.5-turbo', )
system = (
    """You are a soccer scout and you must be good at finding the best
        talents in your team starting from the players rated by the similar
        player system."""
)
system_message_prompt = SystemMessagePromptTemplate.from_template(system)

human_template = """
    EXPLANATION (E) ...

    DATA (D):
    ------------------------------------
    {content}
    ------------------------------------

    FEATURE (F) ...

    SHOW AN EXAMPLE (S) ...
    """

human_message_prompt =
    HumanMessagePromptTemplate.from_template(human_template)
chat_prompt = ChatPromptTemplate.from_messages(
    [system_message_prompt, human_message_prompt]
)

result = chat(
    chat_prompt.format_prompt(
        # Passing parameters at the prompt
    ).to_messages()
)
```

Figure 4.4: Prompt and GPT 3.5 Turbo model - Code Extract

4.4 Evaluation

To evaluate the reports generated by the models for each technique used to construct the prompts, several evaluation methodologies are employed. First, Human Evaluation is used in which 'domain experts' review the generated reports and provide an evaluation based on the completeness, clarity and relevance of the content.
In addition, automated metrics such as Rouge [14] and BERTScore [15] are applied. These metrics provide an objective assessment of the consistency and accuracy of the generated reports by comparing the generated text with the human reference text.

4.4.1 Human Evaluation

The human evaluation process involves domain experts who examine and evaluate the texts generated by the text generation model. In the context of this evaluation, the domain expert assumes the role of evaluator, being aware of the specific requirements and objectives of the system. Several criteria are taken into consideration to evaluate the reports generated by the model, including:

- **Completeness**: Experts assess whether the generated report comprehensively covers the required information and provides sufficient detail to understand the subject matter.
- **Clarity**: Clarity of the report is essential to ensure that information is communicated in an understandable and unambiguous manner. Experts check whether the text is written coherently and uses clear language to convey the information.
- **Relevance of Content**: The relevance of the content in the report to the specific topic or objective is assessed. It is important that the text generated is relevant and appropriate for the context in which it will be used.

Experts assign scores according to these criteria, and the sum of the scores reflects the overall quality of the report generated by the model.

Therefore, a human evaluation in terms of clarity, completeness and relevance of content is shown for each prompt technique used.

- Zero Shot
  - **Completeness** - The report obtained selects a player from the list of player names, justifying the choice on the basis of the team features submitted in the prompt. It does not make any comparisons with other players, nor does it show any statistics on the player or players nominated for recommendation. It does not follow the expected report pattern.
  - **Clarity** - The generated report does not show any ambiguity, it reasons quite well about the player's choice and respects the context in which it is to be inserted but not as expected.
  - **Relevance of Content** - The generated report only shows a single recommended player without making any comparisons with others.
- One Shot
  - **Completeness** - The report generated follows the example given, choosing potential players by justifying the choice on the basis of the team characteristics presented in the request. It does not show any statistics on the player(s) nominated for recommendation.

  - **Clarity** - The generated report shows no ambiguity, also reasons quite well about the player's choice and respects the context in which it is to be placed.
  - **Relevance of Content** - The generated report shows potential players describing their strengths and weaknesses, chooses a potential player and also justifies the choices but the descriptions are gist of the knowledge base the model has.

- Augmented One Shot
  - **Completeness** - The generated report follows the example provided, choosing potential players by justifying the choice based on the team characteristics presented in the request. It shows some statistics on the players proposed for recommendation.
  - **Clarity** - The generated report does not show any ambiguity, reasons quite well about the player's choice and respects the context in which it is to be placed.
  - **Relevance of Content** - The generated report shows potential players describing their strengths and weaknesses, chooses a potential player and also justifies the choices, the descriptions take into account the past data in the prompt.

- Augmented One Shot + Role
  - **Completeness** - The generated report follows the example provided, choosing potential players by justifying the choice based on the team characteristics presented in the request. It shows some statistics on the players proposed for recommendation.
  - **Clarity** - The generated report does not show any ambiguity, reasons quite well about the player's choice and respects the context in which it is to be placed.
  - **Relevance of Content** - The generated report shows potential players describing their strengths and weaknesses, chooses a potential player and also justifies the choices, the descriptions take into account the past data in the prompt.

Model	Prompt	Human Evaluation	Human Rating
GPT 3.5	Zero Shot	Completeness	1/5
		Clarity	3/5
		Relevance of Content	2/5
	One Shot	Completeness	3/5
		Clarity	4/5
		Relevance of Content	3/5
	Augmented One Shot	Completeness	4/5
		Clarity	4/5
		Relevance of Content	4/5
	Augmented One Shot + Role	Completeness	4/5
		Clarity	4/5
		Relevance of Content	4/5

Table 4.2: Human evaluation of prompt techniques

From the evaluations made, it can be seen that the Augmented Generated and Augmented Generated + Role prompts performed better than the other prompt. Note that the following evaluations were made by a series of report generation during the realisation of the recommendation system and during the actual operation of the system.

In particular, reports were also compared with reports generated by the author himself (human references).

However, human evaluation is a valuable method for assessing generative models outputs, **but it can be subjective and prone to bias**. Different human evaluators may have varying opinions, and the evaluation criteria may lack consistency. Additionally, human evaluation can be time-consuming and expensive, especially for large-scale evaluations.

4.4.2 Rouge Evaluation

The ROUGE method (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics used to assess the quality of automatically generated text. These metrics measure the overlap between the generated text and the human reference text to determine how accurately the generated text reflects the intended content. Three of the most common ROUGE metrics are used in the following context and are ROUGE-1, ROUGE-2 and ROUGE-L quantifying precision, recall and their combination F1-score.

4.4.2.1 ROUGE-1

ROUGE-1 is a metric used to assess the quality of automatically generated texts by measuring the overlap between individual words (unigrams) in the generated text and those in the reference text.

Example of unigrams Phrase: ”is a versatile player” $\xrightarrow{unigrams}$ ”is” ”a” ”versatile” ”player”

- **Precision**

$$P = \frac{\text{Number of overlapping unigrams in generated text and reference text}}{\text{Total number of unigrams in the generated text}} \tag{4.7}$$

- **Recall**

$$R = \frac{\text{Number of overlapping unigrams in generated text and reference text}}{\text{Total number of unigrams in the reference text}} \tag{4.8}$$

- **F1-score**

$$F1 = \frac{2 \times P \times R}{P + R} \tag{4.9}$$

4.4.2.2 ROUGE-2

ROUGE-2 is a metric used to evaluate the quality of automatically generated text by focusing on the overlap of word pairs (bigrams) between the generated text and the reference text.

Example of bigrams Phrase: ”is a versatile player” $\xrightarrow{bigrams}$ ”is a” ”versatile player”

- **Precision**

$$P = \frac{\text{Number of overlapping bigrams in generated text and reference text}}{\text{Total number of bigrams in the generated text}} \tag{4.10}$$

- **Recall**

$$R = \frac{\text{Number of overlapping bigrams in generated text and reference text}}{\text{Total number of bigrams in the reference text}} \tag{4.11}$$

- **F1-score**

$$F1 = \frac{2 \times P \times R}{P + R} \tag{4.12}$$

4.4.2.3 ROUGE-L

ROUGE-L is a metric used to assess the quality of automatically generated text, focusing on the longest common sub-sequence (LCS) that overlaps between the generated text and the reference text.

- **Precision**

$$P = \frac{\text{Length of LCS between generated and reference text}}{\text{Total number of words in the generated text}} \tag{4.13}$$

- **Recall**

$$R = \frac{\text{Length of LCS between generated and reference text}}{\text{Total number of words in the reference text}} \tag{4.14}$$

- **F1-score**

$$F1 = \frac{2 \times P \times R}{P + R} \tag{4.15}$$

4.4.2.4 Computing Python

Extract of Python code used for ROUGE:

```
# Import the Rouge scorer module from the Rouge Score package.
from rouge_score import rouge_scorer

# Create a RougeScorer object with specified Rouge metrics ('rouge1',
    'rouge2', 'rougeL')
# and enable the use of a stemmer (typically used for stemming words).
scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'],
    use_stemmer=True)

# Calculate Rouge scores by comparing a reference summary to a candidate
    summary
# and store the results in the 'scores' variable.
scores = scorer.score(reference_summary, candidate_summary)
```

Figure 4.5: ROUGE code used for evaluation of generated texts

4.4.3 BERTScore Evaluation

BERTScore is an automatic evaluation metric used to assess the quality of text generation by comparing the generated text with a human reference text. This metric is named after Google's 'BERT' (Bidirectional Encoder Representations from Transformers) language model.
Unlike previously used metrics such as ROUGE, BERTScore uses a pre-trained language model (BERT) to calculate the semantic similarity between two text sequences. This makes BERTScore more advanced in capturing the semantic meaning of words instead of relying only on exact matches of words or phrases.

- **Tokenisation**: Text sequences (generated text and human reference text) are divided into tokens, which can be words or substrings of words. For example, the sentence 'The cat sleeps' can be divided into the tokens ['The', 'cat', 'sleeps'].

- **Embedding**: Each token is converted into an embedding vector using the BERT model. These embedding vectors represent the semantic meaning of each token in the sequence.
- **Cosine similarity calculation**: For each token in the generated text, the cosine similarity between its embedding vector and the embedding vectors of all tokens in the reference text is calculated. Cosine similarity measures how similar two vectors are to each other.
- **BERTScore P,R and F1 calculation**: cosine similarity scores between generated tokens and reference tokens are used to calculate recall, precision and F1-score.

Illustration (Figure 4.6) of the computation of the recall metric R_{Bert}. Given the human reference x and generated report candidate $\hat{x}$, we compute BERT embeddings and pairwise cosine similarity. We highlight the greedy matching in red, and include the optional idf importance weighting.

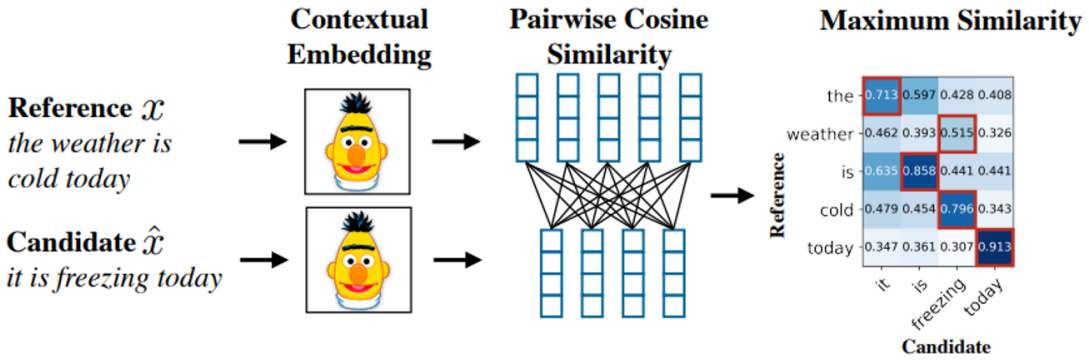

Figure 4.6: BERTScore process [15]

Let be:

- $x = \langle x_1, \dots, x_k \rangle$ is a tokenized human reference, the embedding model generates a sequence of vectors $\langle \mathbf{x}_1, \dots, \mathbf{x}_{ki} \rangle$
- $\hat{x} = \langle \hat{x_1}, \dots, \hat{x_k} \rangle$ is a tokenized generated report, the embedding model generates a sequence of vectors $\langle \hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_k \rangle$
- The cosine similarity of a humane reference token x_i and a generated report token $\hat{x}_j$ is $\frac{\mathbf{x_i^T} \cdot \hat{\mathbf{x}}_j}{\|\mathbf{x_i}\| \|\hat{\mathbf{x}}_j\|}$. We use pre-normalized vectors, which reduces this calculation to the inner product $\mathbf{x}_i^T \cdot \hat{\mathbf{x}}_j$. It's possibile to calculate:

$$R_{Bert} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x_i^T} \cdot \hat{\mathbf{x}}_j \tag{4.16}$$

$$P_{Bert} = \frac{1}{|\hat{x}|} \sum_{\hat{x}j \in \hat{x}} \max_{x_i \in x} \mathbf{x_i^T} \cdot \hat{\mathbf{x}}_j \tag{4.17}$$

$$F_{Bert} = \frac{2P_{Bert} \cdot R_{Bert}}{P_{Bert} + R_{Bert}} \tag{4.18}$$

where $|x|$ and $|\hat{x}|$ are the number of tokens in the reference and candidate sentences, respectively.

Extract code of BERTScore [16]:

```
# Import the 'score' function from the 'bert_score' module in the Hugging
    Face Transformers library.
from bert_score import score

# Calculate precision (P), recall (R), and F1-score (F1) using the 'score'
    function.
# Provide the 'candidate_summary' as the candidate text and
    'reference_summary' as the reference text.
# Set the language to English ('en') and enable verbose mode for more
    detailed information.
P, R, F1 = score([candidate_summary], [reference_summary], lang="en",
    verbose=True)
```

Figure 4.7: BERTScore code used for evaluation of generated texts - Code Extract

4.4.4 Results

The results obtained are shown:

Model	Prompt	Evaluation Metrics	Rouge			BERTScore
			Rouge-1	Rouge-2	Rouge-L	
GPT 3.5	Zero Shot	Precision	0,606	0,235	0,266	0,883
		Recall	0,508	0,191	0,221	0,876
		F1-Measure	0,544	0,207	0,238	0,879
	One Shot	Precision	0,613	0,266	0,301	0,897
		Recall	0,587	0,254	0,298	0,892
		F1-Measure	0,591	0,256	0,295	0,895
	Augmented One Shot	Precision	0,616	0,263	0,329	0,905
		Recall	0,611	0,249	0,327	0,903
		F1-Measure	0,611	0,250	0,326	0,904
	Augmented One Shot + Role	Precision	0,630	0,324	0,378	0,912
		Recall	0,663	0,328	0,328	0,908
		F1-Measure	0,639	0,324	0,381	0,910

Table 4.3: Results obtained

It is important to emphasize that the results were obtained through a meticulous process involving the generation of three reports for each prompt and their comparison with summaries created by the author, followed by the averaging of scores.

First and foremost, it is evident that the addition of content to the prompts has led to an improvement in performance, aligning with expectations. This suggests that providing the model with a greater amount of information

can positively influence the quality of the generated responses.

However, ROUGE was chosen, as the main focus is on the ability of the generative model to cover most of the relevant information present in the human reference text (recall). From the results of ROUGE, it is evident that Rouge-1 scored higher than ROUGE-2 and ROUGE-L, as the higher specificity of the latter two metrics may make it more challenging to achieve high scores. However, ROUGE-2 and ROUGE-L simultaneously represent more precise measures of consistency and relevance in the generated text and assess the exact overlap between human reference and generated report.

Therefore, BERTScore is used to assess the semantic similarity between the generated text and the reference text. The high scores obtained with BERTScore, with an F1-Score of around 90%, indicate that the system is able to generate text that is semantically similar to the human reference text, affirming the effectiveness of generation by the GPT 3.5 model.

However, both ROUGE and BERTScore provide information on the text generation capabilities of the GPT 3.5 model but do not measure the informative quality and fidelity of the generated text. [17]

In light of these analyses, the choice to adopt the Augmented Generated One Shot Prompt with System Role for the final application appears to be a well-founded decision, considering the overall performance achieved.

4.5 Overall methodological architecture of the Player Scouting Recommendation System

The complete architecture of the system is shown by composing all the components previously discussed.

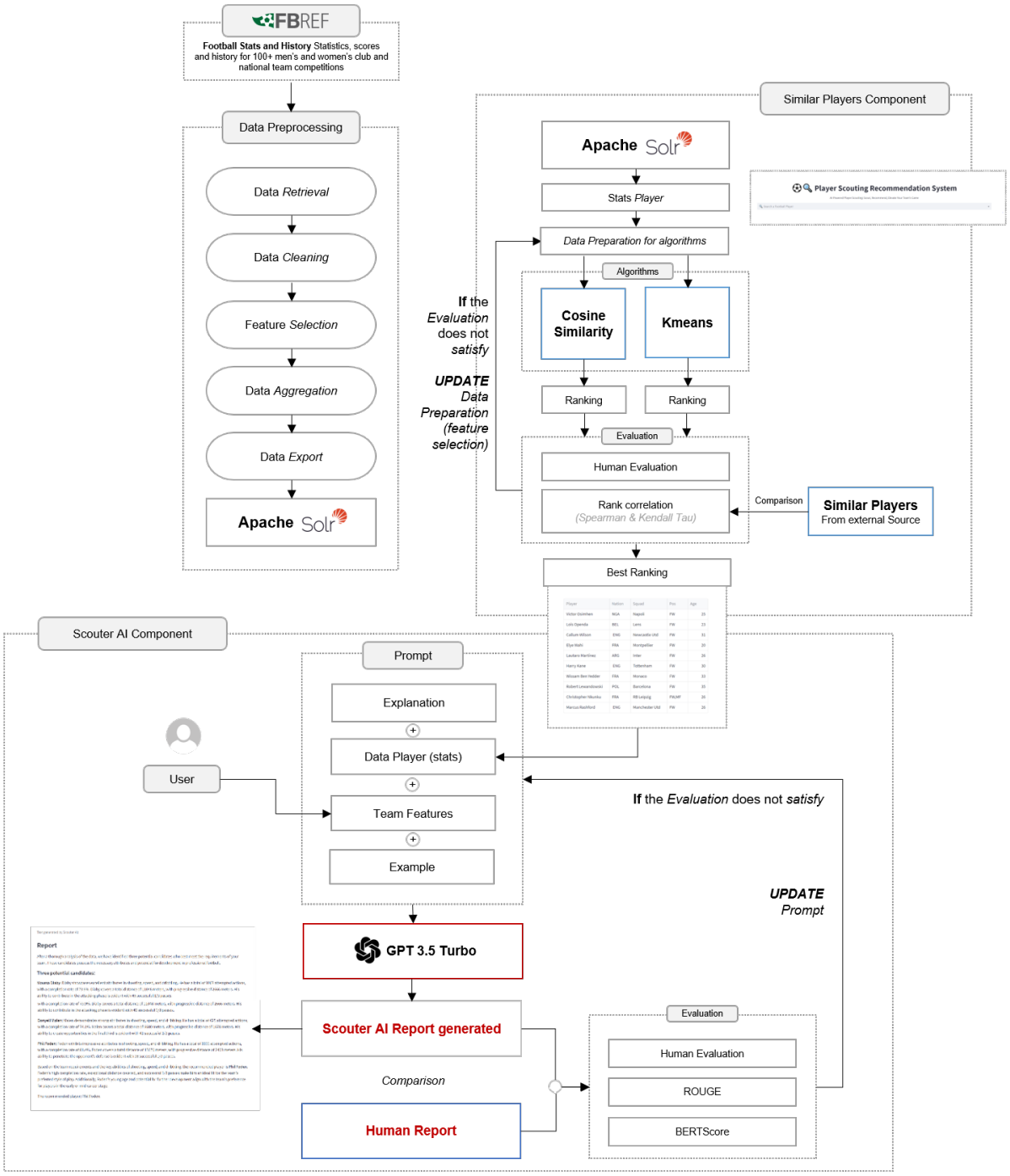

Figure 4.8: Overall methodological architecture of the Player Scouting Recommendation System

5 Conclusion

Chapter Contents

5.1 Application 37
5.2 Pro and Cons 38
5.3 Limits 39
5.4 Future Goals 39

5.1 Application

The entire system, called the 'Player Scouting Recommendation System' [1] , is presented in its entirety, highlighting the operational components. For the realisation of the user interface (UI), the Streamlit framework was employed. The latter was customised to meet the specific requirements of the project, including the integration of a Solr-based search engine to identify players, the implementation of a component dedicated to the evaluation of similar players 5.1, and the use of a generative artificial intelligence model for the generation of scout reports 5.2.

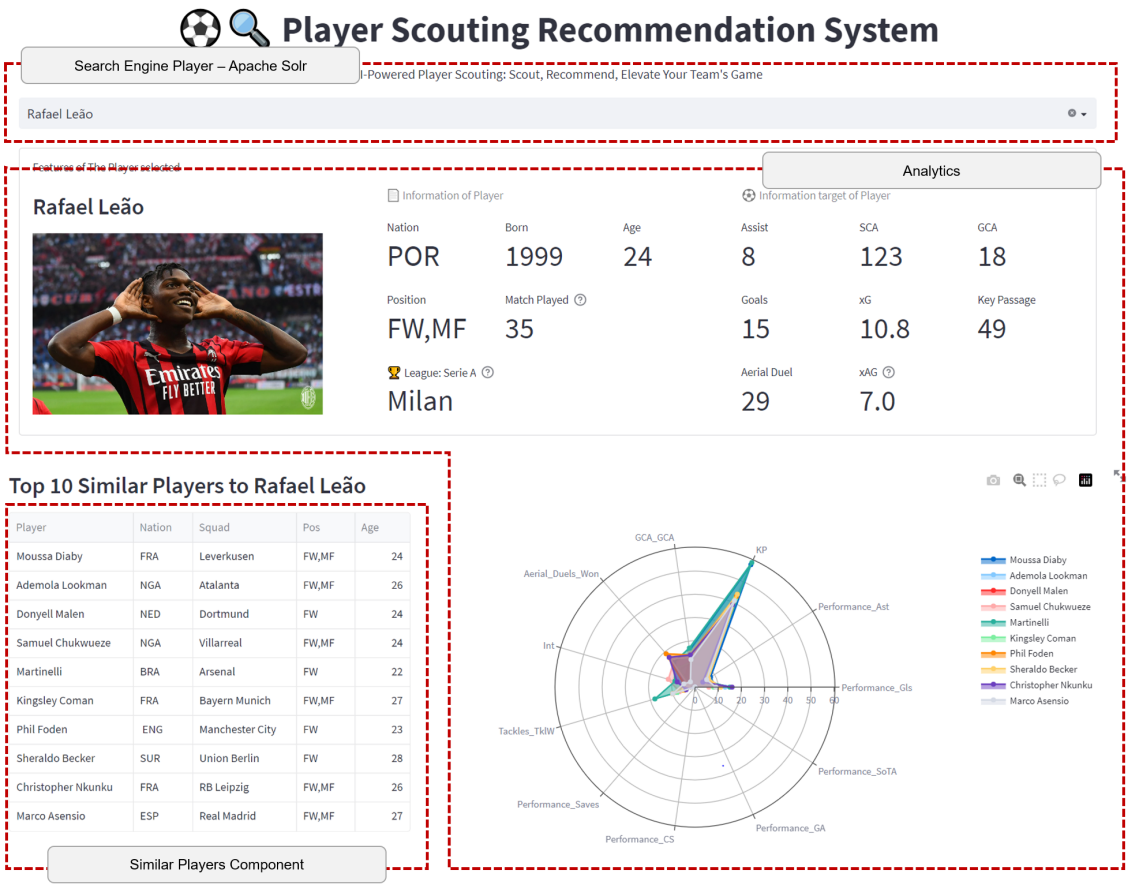

Figure 5.1: Python Application - Simple Search Engine and Similar Player component in Streamlit

[1] The entire code is present in the *attached* **app.py**

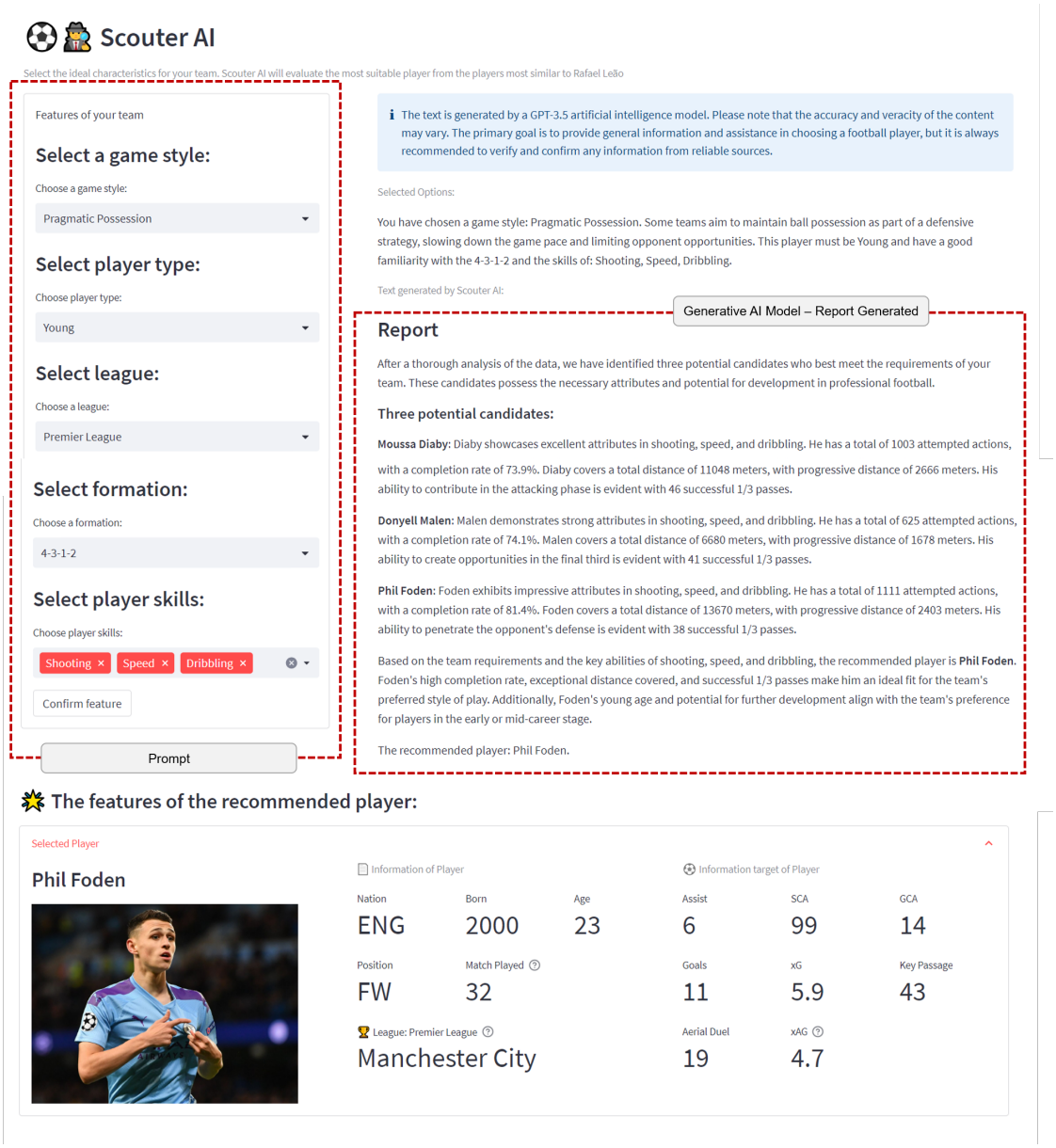

Figure 5.2: Python Application - Fix Form Features Prompt and Report generated

5.2 Pro and Cons

The pros and cons of the system are described:

- Pro:
  - **Decision Support**: The system provides valuable assistance in player selection for professional football teams, especially those competing in the top-5 European leagues
  - **Similarity-Based Recommendations**: The system utilizes a similarity algorithm to identify the top 10 players most similar to the desired ideal profile, streamlining the decision-making process.

  - **Report Generation**: Thanks to the Scouter AI, the system can generate reports on recommended players, offering detailed and insightful information for evaluation.

- Cons:

  - **Temporal Limitation**: The system relies on data updated only within the last year, which may limit the accuracy of recommendations, especially in a context where player performances can change rapidly.
  - **Dependence on Input Data**: The quality of recommendations depends on the completeness and accuracy of input data, meaning that incomplete or erroneous information can negatively impact decisions.
  - **Lack of Context and Human Intuition**: The AI may not consider intangible factors such as team chemistry or player personalities, which could be relevant for purchase decisions.
  - **Technological Limitations**: The accuracy of recommendations is limited by current AI technology and models, which may not fully assess the breadth of a player's skills.
  - **Need for Human Verification**: It is essential for teams to conduct thorough human evaluation before making final player acquisition decisions.

5.3 Limits

The limitations of the project are described:

- **Limitations of the dataset**: The use of a dataset limited to the year 2022/2023 makes the results less reliable, as it does not take into account the entire football career of a player. This limitation may negatively influence the recommendations, especially for those players who have had varying performances over the years. It is important to note that the results of the Player Similarity component were particularly influenced by the amount of matches played during the season. In practice, the more matches a player played, the more accurate the results obtained by the Player Similarity system were.
- **Report generation and hallucinations**: Report generation by a generative AI model can lead to hallucinations or unreliable information. Despite the use of prompt techniques to mitigate this problem, inaccuracies can still occur, calling into question the reliability of generated reports.
- **Limitations of the AI Model**:

  - **Token limitation**: The model has a limitation of input/output tokens, which may result in a limited length of reports generated. In some situations, this restriction may prevent a complete and detailed description of recommended players.
  - **Costs**: The use of generative AI models such as GPT 3.5 can incur significant costs. The author has been limited by a tight budget, and full access to the system requires a paid OpenAI key. This may limit the accessibility of the system to other users as the key is required.

5.4 Future Goals

The future aims of the project are described:

- **Dataset Expansion**: The primary objective is to expand the dataset to include data from multiple different years. This extension will allow for a more accurate representation of players' careers and improve the accuracy of player similarity assessments.
- **Development of Advanced Generative AI Techniques**: Exploring and implementing new Generative AI techniques to mitigate hallucinations and enhance the coherence of generated reports. This may involve adopting more advanced models like GPT-4, LLAMA2, or the upcoming BARD.
- **Prompt Optimization**: Continuously work on optimizing prompts for generative AI models to maximize the coherence and relevance of the generated reports. This could involve identifying new prompt patterns that more effectively guide the text generation process.
- **Collaboration with Industry Experts**: Collaborate with football industry experts to gain a deeper understanding of the needs and expectations of teams and coaches, further tailoring the system to their specific requirements.

List of Figures

1.1 Methodological Workflow - Player Scouting Recommendation System . . . 3

2.1 Workflow - Data Preprocessing . . . 4
2.2 ClienAPP, SolrClient, Apache Solr . . . 5
2.3 Data Export on Apache Solr with SolrClient Library Python . . . 6
2.4 Apache Solr Architecture Scheme for FootballStatsCore . . . 7
2.5 Code extract 'Managed Schema' file for the configuration of the Apache Solr Core - FootballStatsCore 8
2.6 Autocomplete system implemented with streamlit component and SolrClient . . . 8
2.7 Solr with Streamlit autocompletation - Code Extract . . . 9

3.1 Workflow - Similar Players Component . . . 10
3.2 Cosine Similarity - Mathematical representation . . . 11
3.3 Cosine Similarity for Similar Players - Code Extract . . . 12
3.4 Cosine Similarity Matrix . . . 13
3.5 Elbow Method for optimal value of k in KMeans . . . 15
3.6 Kmeans and Elbow method for selection K - Code Extract . . . 16
3.7 Calculation of Silhouette Score and Ranking - Code Extract . . . 18
3.8 Spearman and Kendall Tau - Code Extract . . . 20

4.1 Methodological workflow of Scouter AI component . . . 24
4.2 Combination of Prompt . . . 25
4.3 Generated Report Example . . . 27
4.4 Prompt and GPT 3.5 Turbo model - Code Extract . . . 28
4.5 ROUGE code used for evaluation of generated texts . . . 32
4.6 BERTScore process [15] . . . 33
4.7 BERTScore code used for evaluation of generated texts - Code Extract . . . 34
4.8 Overall methodological architecture of the Player Scouting Recommendation System . . . 36

5.1 Python Application - Simple Search Engine and Similar Player component in Streamlit . . . 37
5.2 Python Application - Fix Form Features Prompt and Report generated . . . 38

List of Tables

3.1 Ranking evaluated by Cosine Similarity - Example on Erling Haaland . . . 13
3.2 Silhouette Score . . . 17
3.3 Ranking evaluated by Kmeans - Example on Erling Haaland . . . 18
3.4 Ranking from Kmeans . . . 19
3.5 Ranking from Cosine Similarity . . . 19
3.6 Example of Erling Haaland . . . 19
3.7 Rank of top 10 similar players from **FBRef** . . . 20
3.8 Other Results of Spearman and Kendall for each algorithm used . . . 21

4.1 Choice of generative models . . . 23
4.2 Human evaluation of prompt techniques . . . 30
4.3 Results obtained . . . 34

Bibliography

[1] FBref. *FBref - Football Statistics and History*. 2023. URL: https://fbref.com/en/.

[2] Apache Solr. *ASCII Folding Filter*. 2023. URL: https://solr.apache.org/guide/6_6/filter-descriptions.html#FilterDescriptions-ASCIIFoldingFilter.

[3] Abhinav Agarwal. *Cosine Similarity: How does it measure similarity? Math behind and usage in Python*. 2023. URL: https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db.

[4] Khyati Mahendru. *How to Determine the Optimal K for K-Means?* 2023. URL: https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb.

[5] Ashutosh Bhardwaj. *Silhouette Coefficient: How does it measure similarity? Math behind and usage in Python*. Ed. by Towards Data Science. July 20, 2023. URL: https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c.

[6] Aditya Aggarwal. *Spearman Correlation Coefficient: Formula, Examples, and Interpretation*. 2023. URL: https://vitalflux.com/spearman-correlation-coefficient-formula-examples/.

[7] AIMultiple. *AI Text Generation*. 2023. URL: https://research.aimultiple.com/ai-text-generation/.

[8] AIMultiple. *Large Language Models*. 2023. URL: https://research.aimultiple.com/large-language-models/.

[9] Prompting Guide. *AI Text Generation*. 2023. URL: https://www.promptingguide.ai/it/.

[10] KDnuggets. *Packt Summary of GPT-3*. 2022. URL: https://www.kdnuggets.com/2022/04/packt-summarization-gpt3.html.

[11] Prompting Guide. *Zero-Shot Text Generation*. 2023. URL: https://www.promptingguide.ai/it/techniques/zeroshot.

[12] Prompting Guide. *Few-Shot Text Generation*. 2023. URL: https://www.promptingguide.ai/it/techniques/fewshot.

[13] Amogh Agastya. *Harnessing Retrieval Augmented Generation With Langchain*. 2023. URL: https://betterprogramming.pub/harnessing-retrieval-augmented-generation-with-langchain-2eae65926e82.

[14] Ahmed Waheed. *How to Calculate Rouge score in Python*. June 2023. URL: https://thepythoncode.com/article/calculate-rouge-score-in-python.

[15] Tianyi Zhang* et al. "BERTScore: Evaluating Text Generation with BERT". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SkeHuCVFDr.

[16] *BERTScore - Google Colaboratory*. 2020. URL: https://colab.research.google.com/drive/1kpL8Y_AnUUiCxFjhxSrxCsc6-sDMNb_Q.

[17] Daniel Deutsch and Dan Roth. *Understanding the Extent to which Summarization Evaluation Metrics Measure the Information Quality of Summaries*. 2020. eprint: arXiv:2010.12495.

[18] SolrClient Team. *SolrClient Documentation*. 2023. URL: https://solrclient.readthedocs.io/en/latest/.

[19] AIMultiple. *GPT*. 2023. URL: https://research.aimultiple.com/gpt/.