Data Mining 2021/22 Project Report

Università di Pisa, Department of Computer Science

Tennis Matches

Dininta Annisa (626660) – d.annisa@studenti.unipi.it
Andrea Laretto (619624) – a.laretto@studenti.unipi.it
Pier Paolo Tarasco (619622) – p.tarasco@studenti.unipi.it

Contents

1 Data Understanding

In this chapter we present a general overview of input datasets involved in the project, assessing their quality and explaining our solutions to the most relevant issues present on the data. Additionally, we introduce the semantics of the indicators of the datasets not explained in the project description. The datasets provided for the task are:

- `male_players.csv` (*55208 rows*),
- `female_players.csv` (*46172 rows*),
- `tennis_matches.csv` (*186128 rows*).

Both `male_players.csv` and `female_players.csv` datasets are composed by 2 `object`-type attributes **name** and **surname**, while `tennis_matches.csv` is composed of **49 attributes**, of which **35 numerical** and **14 categorical**.

1.1 Data semantics

We provide here a brief explanation on the remaining attributes not described in the project specification. We were able to document these attributes by both researching about the problem domain and by experimenting with the attribute values themselves.

- **score** (`object`, *85929 non-null*): space-separated scores for each set of the match, with scores separated in winner and loser by a dash (e.g.: `6-3 6-4`, and `4-6 6-3 6-4`). The entire score can also be `W/O` in case of a walk-out, and similarly a set score of `RET` indicates that one of the two players retired. If a score of `6-6` is reached in a set, a tie-breaker game is played and the loser's score is indicated in round brackets next to the set score, which is either `6-7` or `7-6` depending on the winner (e.g.: `6-7(3)`, and `7-6(1)`). Finally, the last set can be shown in square brackets to indicate a final tie-break set, where 10 points are required to win (e.g.: `6-1 6-7(2) [10-4]`).
- **round** (`object`, *86098 non-null*): indicates the round of the match inside the tournament. The possible values are:
  - `R16`, `R32`, `R64`, `R128` (rounds of 16, 32, 64, or 128),
  - `Q1`, `Q2`, `Q3` (first, second, and third qualification rounds),
  - `QF`, `SF`, `F` (quarter-finals, semi-finals, and finals),
  - `RR` (matches played in tournaments based on round-robin).
- **winner_name** and **loser_name** (`object`, *186101 non-null*): full name of the winner/loser, usually composed of the concatenation between name and surname (even for asian onomastics, e.g. Japanese and Korean surnames)
- **tourney_date** (`float64`, *186100 non-null*): date of the match, indicated with a YYYYMMDD format with a floating point value.
- **tourney_spectators** (`float64`, *186101 non-null*): number of spectators present at the tournament match.
- **tourney_revenue** (`float64`, *186102 non-null*): revenue of the tournament.

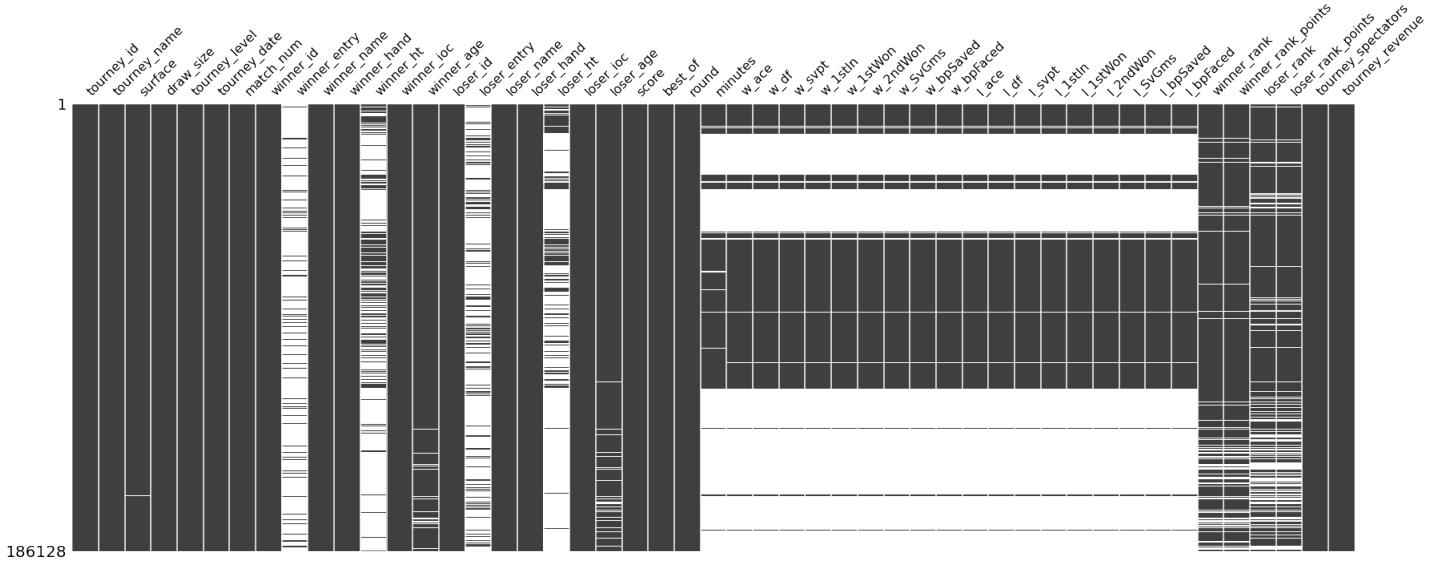


Figure 1: `NaN` values in `tennis_matches`, missing values in rows are shown in white.

1.2 Data quality

We report in Figure 1 the proportion of `NaN` values in each row of `tennis_matches`. Aside from the presence of `NaN` in match-related data, we address several problems we discovered related to the tennis players:

- From the total 101380 players (by combining `male_players` and `female_players`) we dropped 42 rows which had both `NaN` name and surname along with 994 duplicate rows.
- 1696 of the players contained `NaN` in either name or surname, which were not dropped but for which the `full_name` consists of just the non-null attribute without concatenating them.
- The full names of 30 players appear in `tennis_matches` but are not found in either `male_players` or `female_players`. By also consulting external information, we discovered that 29 of these were simply typos, while only 1 record was missing. We manually fixed the errors and added the missing player since they played 70 matches.
- 6 player names from `tennis_matches` appear in both `male_players` and `female_players`. To determine the real gender, we checked the opponent's gender. We also discovered that one of them is actually 2 different players with different gender, which we took care into not dropping them.
- 97 matches are missing both `full_name` and `player_id` in either winner or loser; the matches were not dropped and the statistics of the other player were preserved for the profile.
- 93 rows contained `NaN` values in the `hands` attribute of `tennis_matches`. Since `U` is also used for unspecified as specified in the project description, we aligned all unspecified values to `NaN`.
- It turns out that the `player_id` in `tennis_matches` does not uniquely identify the players, i.e.: 49 players appear with same `id` and different name (at most 2 people), while 19 players appear with same full name with different `id`s. We checked with external sources that indeed there is only one high-profile tennis player for each of these names. We then considered the combination of full name and gender as unique and we modified the `id` by creating an entirely new incremental `id` which we later replaced in all relevant datasets.

Attribute	Mathematical formulation
Percentage of both return/serve points won	$\texttt{w\_ptWon\%} = \frac{\texttt{w\_1stWon}+\texttt{w\_2ndWon}-\texttt{l\_1stWon}-\texttt{l\_2ndWon}+\texttt{l\_svpt}}{\texttt{w\_svpt}+\texttt{l\_svpt}}$
Percentage of first serves executed	$\texttt{1st\%} = \texttt{1stIn}/\texttt{svpt}$
Percentage of first serve points won	$\texttt{1stWin\%} = \texttt{1stWon}/\texttt{1stIn}$
Percentage of valid second serves	$\texttt{2ndIn} = \texttt{svpt} - \texttt{1stIn} - \texttt{df}$
Percentage of second serve points won	$\texttt{2ndWin\%} = \texttt{2ndWon}/\texttt{2ndIn}$
Percentage of aces over all serves	$\texttt{ace\%} = \texttt{ace}/\texttt{svpt}$
Percentage of double faults over all serves	$\texttt{df\%} = \texttt{df}/\texttt{svpt}$
Percentage of break points saved	$\texttt{bpSaved\%} = \texttt{bpSaved}/\texttt{bpFaced}$
Percentage of service points won	$\texttt{svptWon\%} = (\texttt{1stWon} + \texttt{2ndWon})/\texttt{svpt}$

Table 1: Custom attributes defined for `match_side`.

2 Data Preparation

In this chapter we present the main transformations applied to the dataset, the definitions of new indicators and their meaning, and show the resulting histograms and statistics on the datasets.

As one of the first pre-processing operations we performed on the dataset, we merged the `male_players` and `female_players` in a single `all_players` dataframe while adding a `sex` attribute. Additionally, we splitted the winning and losing side of `tennis_matches` and stored it in a new dataframe `match_side`. We added a boolean `win` attribute to indicate which row correspond to the winner and loser, and an unique `match_id` so that the entire match can eventually be reconstructed with a self-join and without loss of information. This operation greatly simplified the processing of the dataset and allowed us to uniformly work with the losing and winning sides and their statistics, as well as simplifying the count of won and lost games for each player.

2.1 Missing values

As we have shown in Figure 1, out of the 186128 matches only 82198 have all non-null values for the most statistically relevant attributes for matches (e.g.: `ace`, `df`, `1stIn`, `2ndWon`, `bpSaved`, etc.), with their validity being highly correlated since very frequently they are all null or all present. Because of their relevance, we decided against using interpolation methods to fill them and we have to resort to drop these null records. However, we still extract as much information as possible *before* dropping the rows, e.g. computing the players' win rate.

2.2 Outliers

To analyze outliers, we plotted the distribution of each attribute of `match_side`, as reported in Figure 2. This revealed the presence of many outliers in the majority of attributes. We tried removing them by dropping values outside the interquartile range, but this did not produce satisfactory results, since high-calibre players were often being excluded because of their uncommon statistics (e.g.: on `rank_points`). Therefore, we instead decided to manually determine the cutoff threshold at reasonable values for each individual attribute. In total, we remove 1213 matches.

2.3 Definition of match indicators

We defined several new attributes in `match_side`, which we aggregated to construct the player profiles and their indicators. We list their description and mathematical definition in Table 1.

2.4 Data statistics and distributions on matches

We show the distribution for each attribute of `match_side` in Figure 2 and the boxplots in Figure 3. The red dashed lines indicate the threshold where we cut off the outliers.

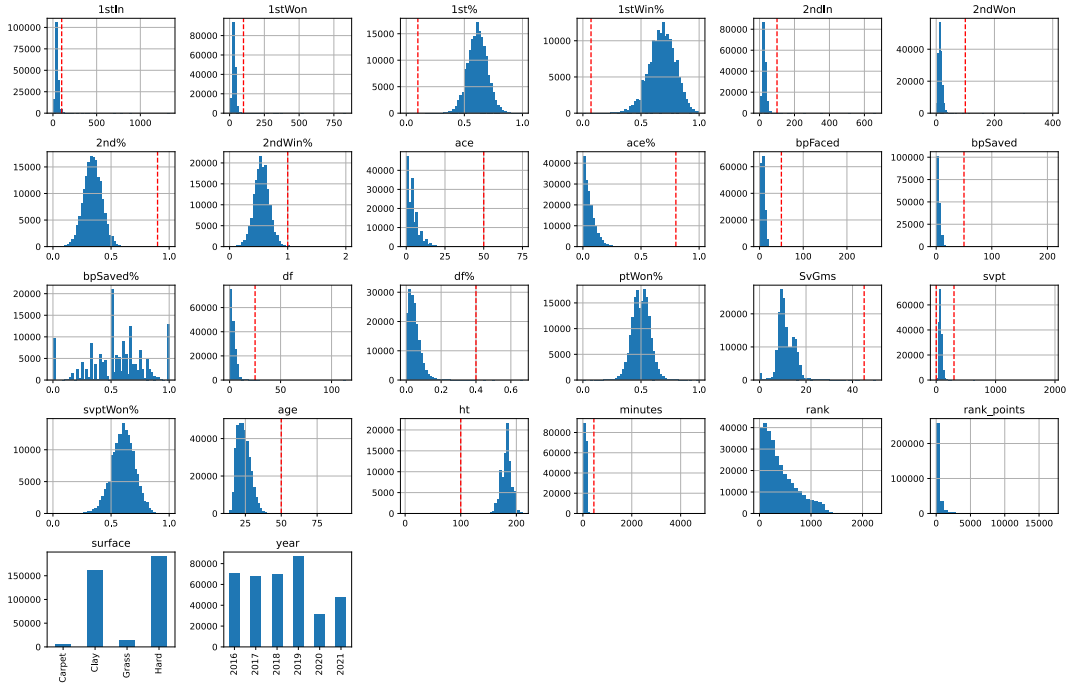


Figure 2: Histograms of the attributes of `match_side`, with thresholds shown as red dashed lines.

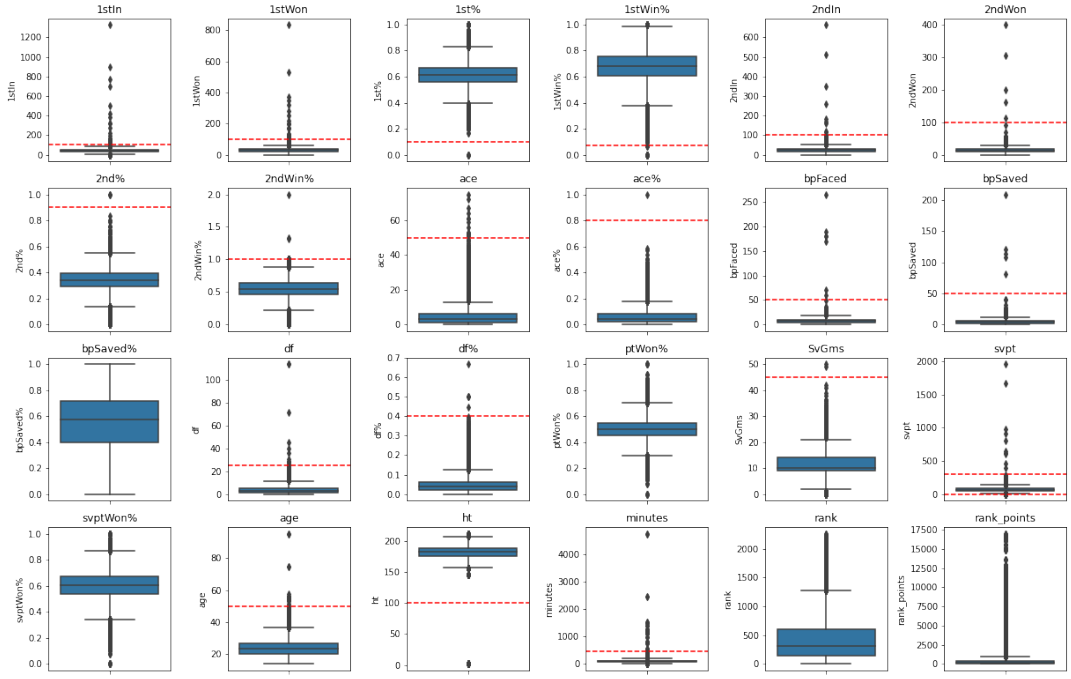


Figure 3: Boxplots of the attributes of `match_side`, with thresholds shown as red dashed lines.

2.5 Definition of player indicators

From the information in `match_side`, we computed the new indicators of each player as follows:

- `n_matches`: the total number of matches a player participated in.
- `n_wins`: the total number of matches won by each player.
- `avg_aces`, `avg_ace%`, `avg_dfs`, `avg_df%`, `avg_1st%`, `avg_1stIn`, `avg_1stWin%`, `avg_2ndIn`, `avg_2nd%`, `avg_2ndWin%`, `avg_bpSaved%`, `avg_svptWon%`, `avg_ptWon%`, `avg_minutes`: the average of the corresponding attribute across all matches played by a player.
- `hand`, `ioc`: these attributes are simply extracted from the matches.
- `age`, `rank`, `rank_points`: these attributes might vary in different matches over different periods of time. We decided to take the median age as representative of the player's age, while taking the minimum rank and maximum rank points to express the maximum potential reached by a player.
- `grass`, `clay`, `hard`, `carpet`: the number of times a player played in a certain kind of surface.
- `grassWins`, `clayWins`, `hardWins`, `carpetWins`: the total number of matches won by each player on a certain kind of surface.
- `total_R16`, `total_R16`, `total_R8`, `total_R4`, `total_R2`, `total_R1`: the number of times a player plays in the corresponding round.

2.6 Statistically relevant player profiles

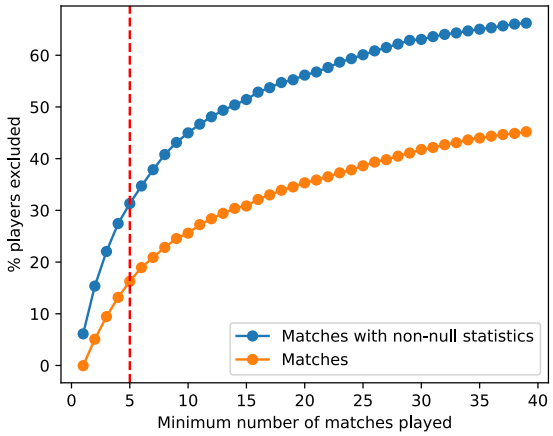


Figure 4: Percentage of players removed by min. number of matches (with statistics) played.

We remark here that among the total 100344 players available in `all_players`, only 10123 players appear in at least one match. Furthermore, since the players indicators are mainly computed by taking averages and sums over the matches played, we believed it is reasonable to exclude from the analysis the players which only played a very few number of matches, so that the cumulative indicators have a statistically meaningful value. For this reason, we experimented with different thresholds of *minimum number of matches played with non-null statistics*, thus excluding the players with too few matches with statistics. At the cost of working on a smaller dataset and excluding minor players, this noticeably improved the clustering quality. As a compromise between dataset size and increasing the quality of our statistics we settled on removing the players with **less than 5 matches**, obtaining the final number of **1918 players profiles**.

2.7 Data statistics on player profiles

In Figure 5 and Figure 6 we show the distribution for the main attributes of `player`, along with their respective correlations in Figure 7. These will be later analyzed to decide the attributes on which to cluster, since non-correlated attributes typically improve the clustering results. For brevity, we leave the boxplots in the notebook and only report in Figure 5 the players histograms.

We briefly comment here on the overall statistics and distributions obtained for the players profiles. The most important tennis statistics related to the serve clearly seem to follow a normal distribution, as is the case for, e.g., `avg_1st%`, `avg_1stWin`, `avg_2nd%`, with occasionally skewed cases such as `avg_ptWon%`, `avg_aces`, and `avg_df`. On the other hand, the global statistics on all the profiles such as `n_matches`, `rank_points`, and the tournament round participations, seem to follow a skewed distribution. We also remark that a small percentage of players plays the majority of matches. We finally note how the male sex is overrepresented in the dataset by more than twice the amount of female players, with the right hand clearly being the most common.

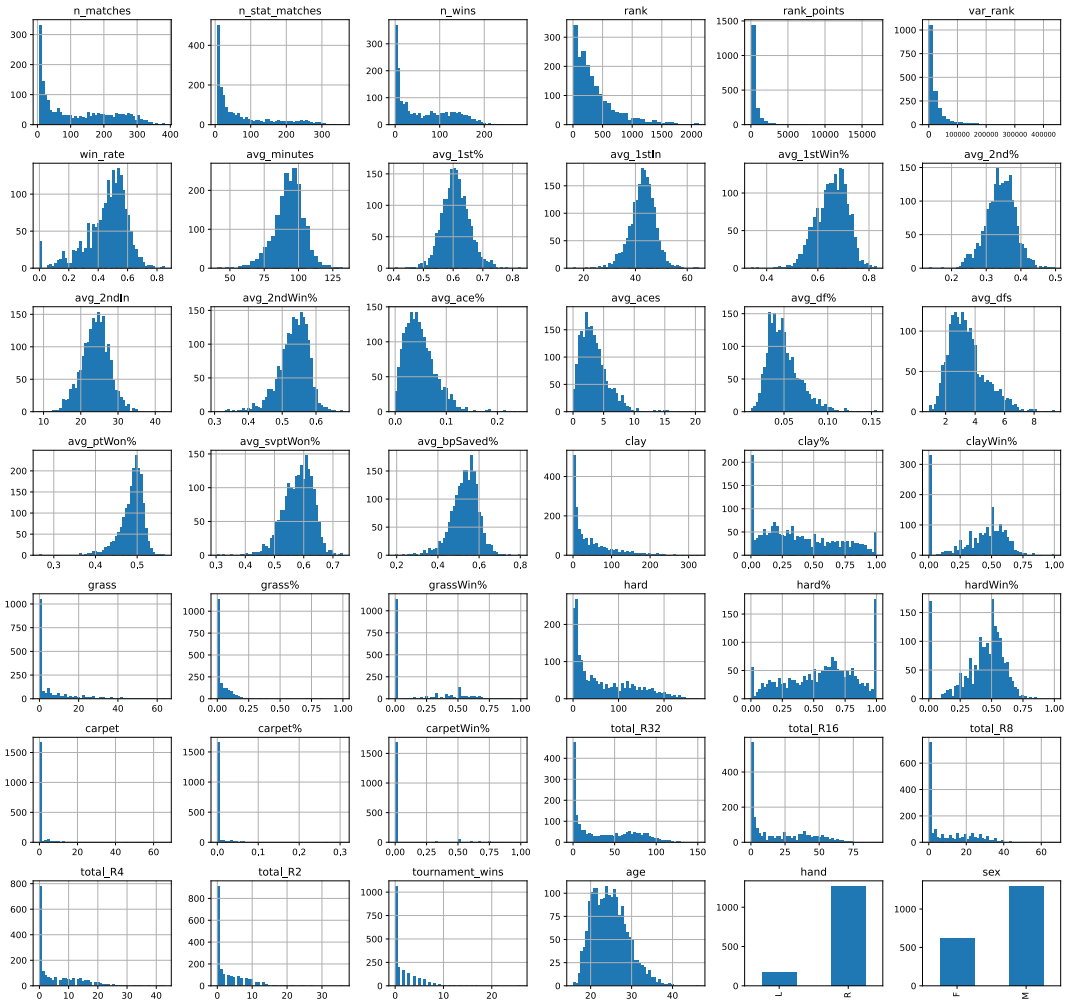


Figure 5: Histograms of the 1918 players profiles after removing outliers.

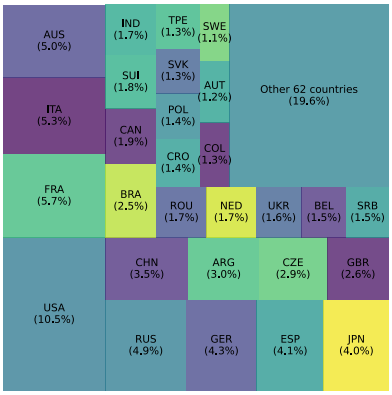


Figure 6: Country distribution of the 1918 players profiles.

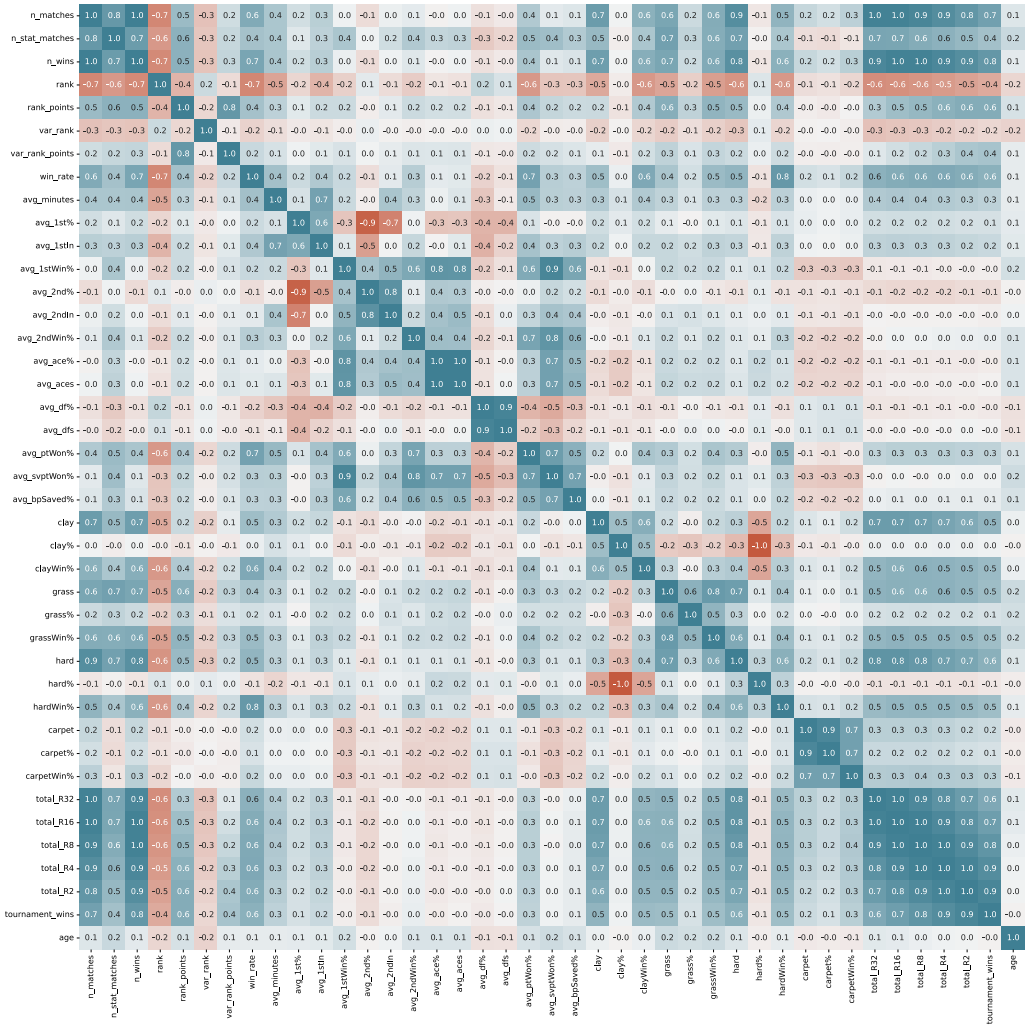


Figure 7: Correlations of the attributes of `players`.

3 Clustering

In this chapter, we present several clustering analyses we performed on the cleaned dataset and we analyze the results obtained along with their performance evaluation.

Methodologically, we first started by identifying a good set of attributes on which to apply the clustering. Then, after running each clustering algorithm, we validated its performance by looking at the `SSE`, `Silhouette`, and `Davies-Bouldin` score of the resulting cluster assignment. Finally, for each result, we analyzed the boxplots of all relevant attributes divided by cluster. We expected a good clustering to be easily distinguishable and produce different distributions for all the clustering groups.

3.1 Attributes selection

In order to find a good combination of attributes to cluster on, we tried to focus on a combination of both tennis-relevant statistics, e.g. `avg_ace%`, `avg_1stIn%`, etc., along with general statistics related to the player's performance such as `win_rate`, and `rank_points`. In order to improve the clustering, we searched for a set of attributes with pairwise correlation at least less than 0.5, which we verified using the correlations in Figure 7. We limited ourselves to a relatively small but significant number of attributes to reduce the effects of the curse of dimensionality and to increase the performance scores. In the end, the attributes we settled on to use in clustering are `avg_ace%`, `avg_df%`, `avg_1st%`, `avg_bpSaved%`, `win_rate`, and `rank_points`, since this set produced the best result with distinguishable clusters. Before clustering, we normalized these attributes using `StandardScaler`, which, especially on k-means, gave experimentally better results compared with other scalers such as `MinMaxScaler` and `RobustScaler`.

3.2 K-means

In order to select the number of clusters k, we considered the *Elbow method* by plotting the `SSE`, `Silhouette` and `Davies-Bouldin` score for k up to a limit, as can be seen in Figure 8. The elbow method heuristically suggests to pick a k as close as possible to the elbow of the `SSE` score, while also trying to maximize the `Silhouette` value and minimize the `Davies-Bouldin` score.

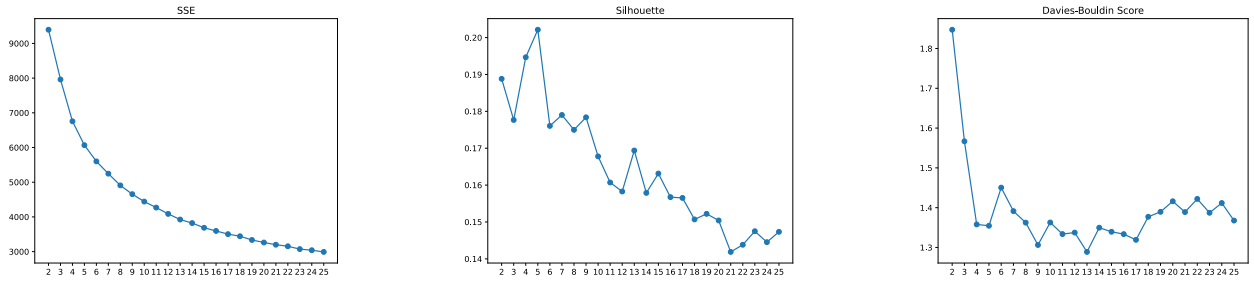


Figure 8: Plots of the SSE, Silhouette score, and Davies-Bouldin score of k-means.

Despite the elbow method heuristically suggesting a larger value of k, we settled on $k = 5$, as it fittingly corresponds to the global maximum value of the Silhouette and a local minima for the Davies-Bouldin score. By looking at the trend of the above mentioned scores, we also considered 9 and 13 as possible values of k; these cluster sizes unfortunately provided very chaotic and hard to interpret cluster characterization, which only further subdivided the clusters without providing additional insight. The cluster sizes can be seen in Figure 9, while the cluster distributions on the main numerical attributes are presented in Figure 10 and Figure 11. Based on the attribute distributions, we extrapolated the following descriptions and interpretations for each cluster:

- Purple (80 players) [*Professionals*]: the best players in the field, with extremely high `win_rate` and `rank_points`. They also tend to play longer games, and have good statistical values all over the board, with `avg_ptWon%` as one of their most distinguishing statistic. We notice how even high rank players have an average `avg_1st%` and `avg_2nd%`, with slightly more frequent successful first serves `avg_1stIn`, possibly indicating that these statistics do not play an essential role at the top level. They are all-courters and excel on every kind of surface as can be witnessed by their win rates, but overall have a preference for the more common `hard%` terrain.

- Red (273 players) [*Occasionals*]: the worst players overall. This can be seen from their low `win_rate` and high `rank` position. In particular, they play very few matches, which is reasonable since in tournaments the winners play more matches. Their first serve `avg_1st%` and `avg_1stIn` is one of the weakest one, with a high number of double faults `avg_df%`, while in matches they correspondingly have the the lowest percentage of points won `avg_ptWon%` and break points saved `avg_bpSaved%`.

- Orange (520 players) [*Safe hitters*]: a common play of style characterized by one of the best first serves `avg_1st%`, and consequently by very unfrequent `avg_2ndIn`. Their playstyle seems to aim at safely securing the first serve without risks, as can be seen by the low number of aces `avg_ace%` and low double faults `avg_df%`. Furthermore, they play the highest percentage of matches on `clay%`, and we speculate that this kind of softer surface is more apt to their safer playstyle.

- Blue (705 players) [*Standard players*]: the most common player profile, with the highest number of players. Has the second lowest number of matches played, with a slight preference for matches with `hard%` surface instead of `clay%`. Their playstyle is characterized by quite good `avg_1stWin%` and `avg_2ndWin%` on both serves, with the second serve `avg_2nd%` being more common. Compared with the *Safe hitters* cluster, they also have a low number of double faults `avg_df%`, but the *Standard player* has a considerably higher `avg_ace%` and `avg_svptWon%`, which is comparable with the high rank *Professionals* players.

- Green (340 players) [*Return hitters*]: players with a unique playstyle with average serve statistics and a high `win_rate` and `n_matches`, both the highest among non-professional players. Surprisingly, they are characterized with the highest percentage of double faults `avg_df%` among all profiles, which seemingly contradicts their high win rate: this might indicate that they play considerably better at return, which does not provide them good serve statistics and instead places them comparable with *Standard players*; despite their low overall statistics, their playstyle still allow them to place as one of the best player class in tournaments positions, e.g.: `total_R8`, `total_R4`, `total_R2`, etc.

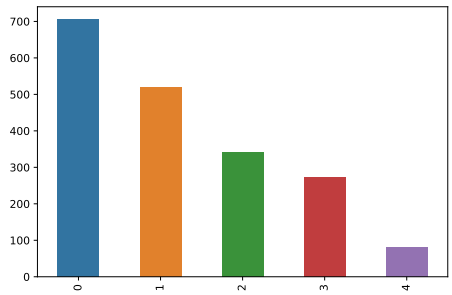


Figure 9: Sizes of the clusters for k-means.

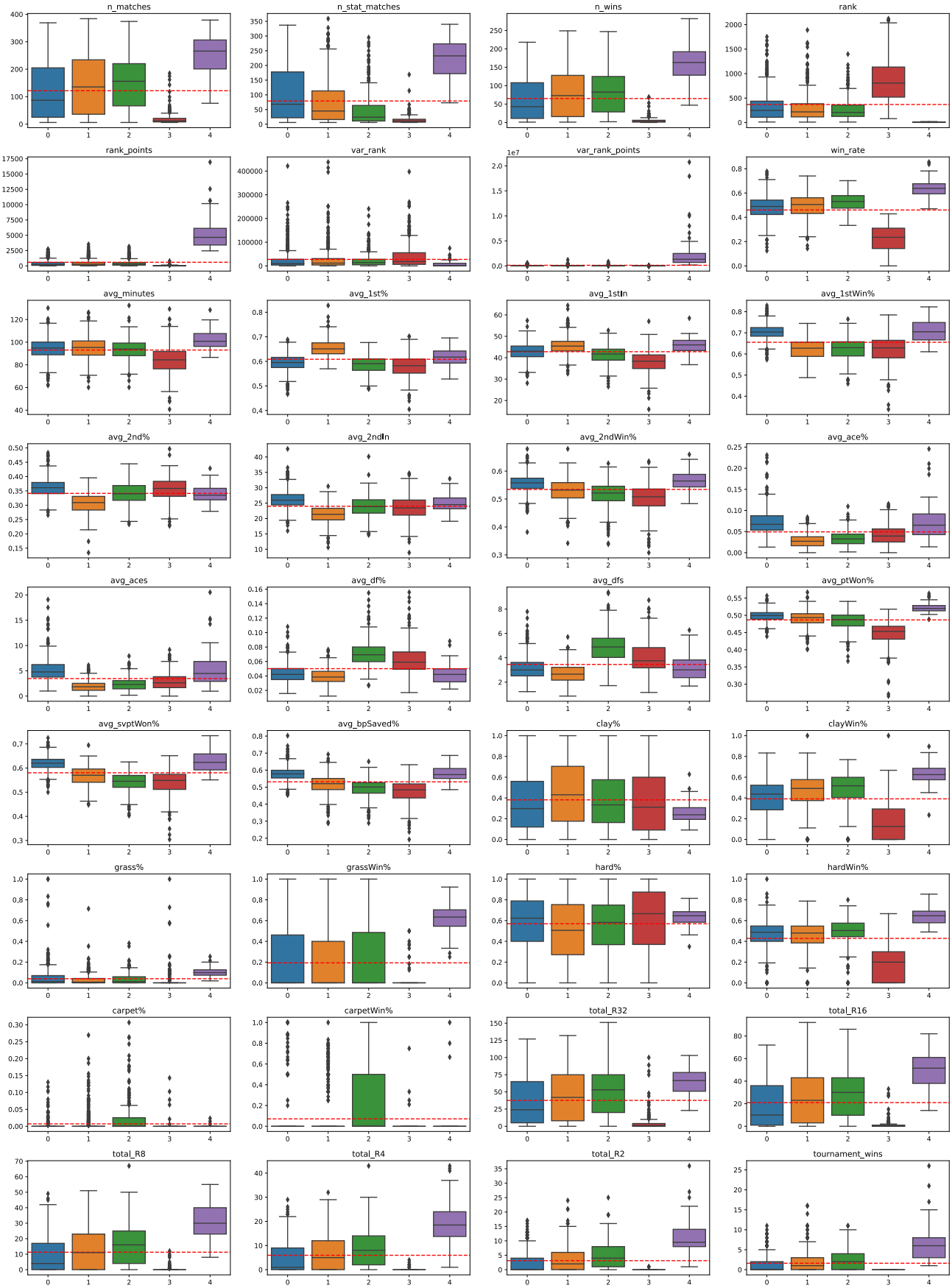


Figure 10: Boxplots of k-means clustering; the red dashed lines indicate the attribute mean.

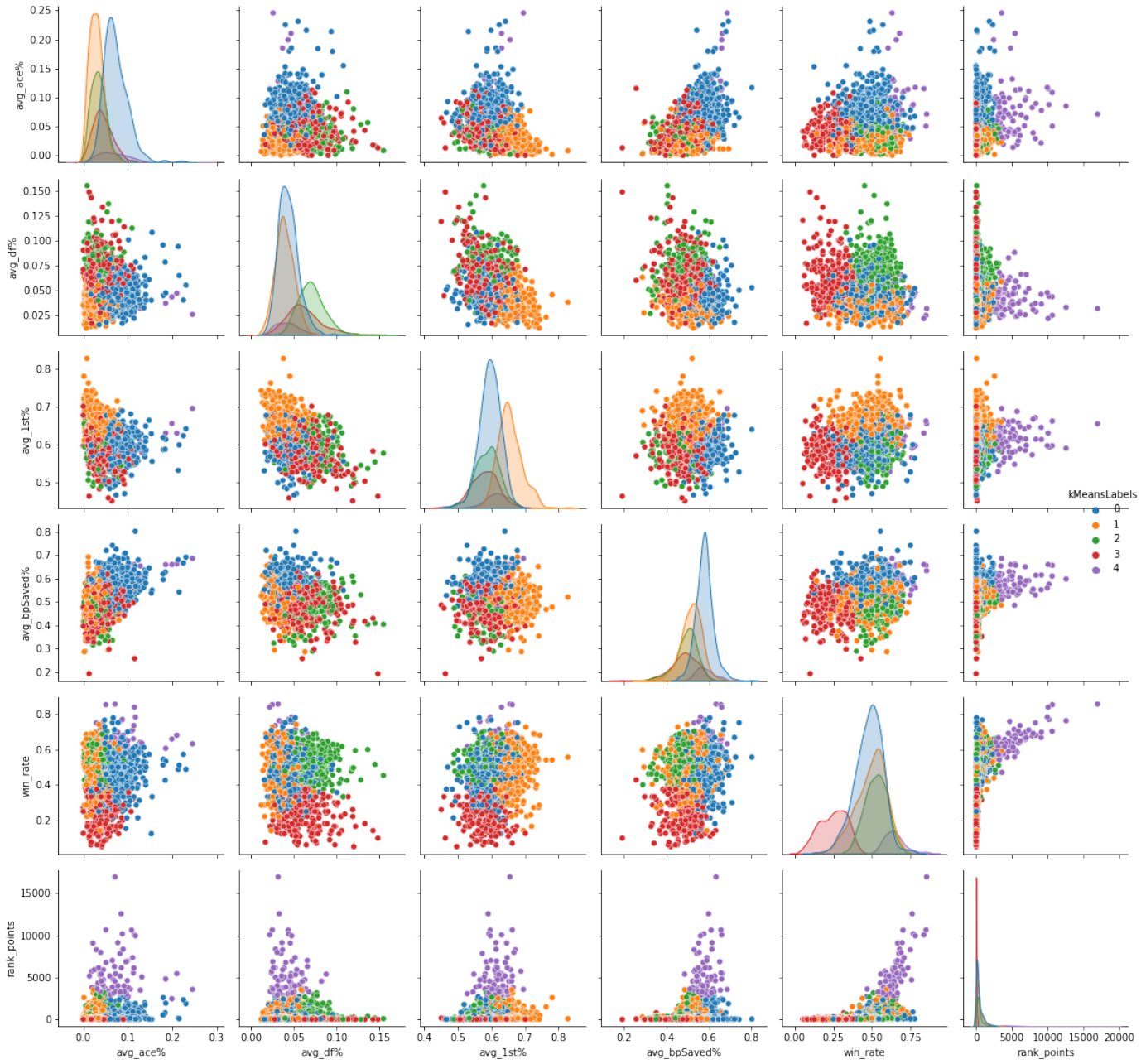
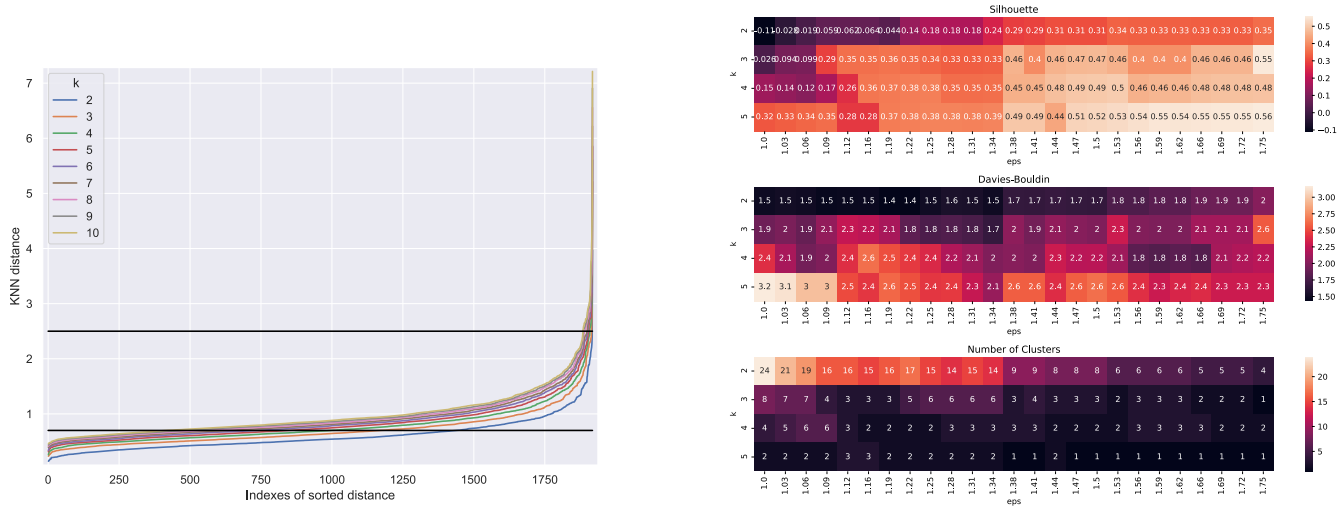


Figure 11: Scatter plots for the cluster subdivisions of k-means.

3.3 DBSCAN

We used the knee method to determine the parameter values of DBSCAN (`eps` and `min_samples`). This is done by computing the k-NN distances of data samples with k in range [2,10], sorting the distances in ascending order, and plotting them as shown in Figure 12a. Then, we searched the optimal value of `eps` and `min_samples` in the "knee" range of the plot, which we marked by two horizontal black lines in Figure 12a. The result of this grid search is shown in Figure 12b. We only report for k in range [2, 5] and `eps` in range [1, 1.75], since values outside this range produce either a single big cluster, negative Silhouette score, or high Davies-Bouldin score.



(a) Sorted k-NN distances between points

(b) Grid search results

Figure 12: Plot of data distances and grid search results for DBSCAN.

We would like a clustering result with high Silhouette score and low Davies-Bouldin score. At the same time, the number of non-noise clusters should be more than 1 in order to obtain a meaningful partitioning of the data. One of the configurations which offer the best compromise seem to be `eps` = 1.41 and `min_samples` = 3. However, these parameters produce imbalanced clusters with (1858, 5, 3, 3) players, while the remaining 49 players are considered as noise. Changing these parameters did not significantly improve the result. The most likely explanation is that the data points are concentrated in one area, so the algorithm sees it as one dense cluster and cannot meaningfully distinguish it.

To better verify this, we also plotted the data in 3 dimensions by considering some selections of 3 attributes out of 6 attributes we clustered on, which we report in Figure 13. We can notice how some points are outside the green data blob, which represent the very best and worst players. However, their high distance from each other makes it so that they are labeled as outliers and cannot form a single well-characterized cluster.

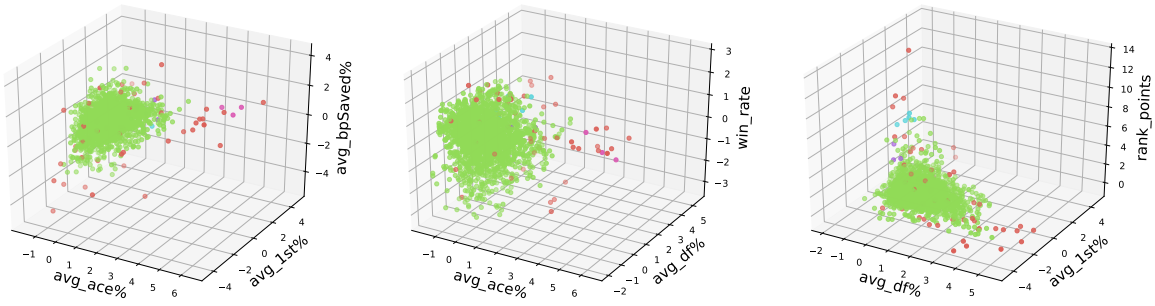
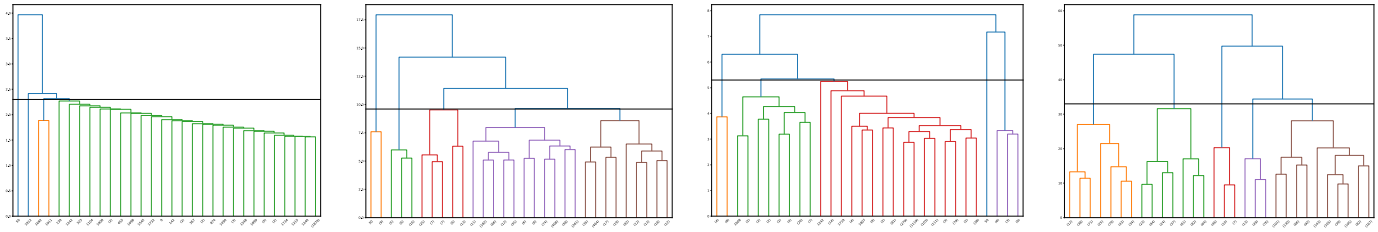


Figure 13: Cluster distribution of DBSCAN, visualized in 3 dimensions.

3.4 Hierarchical clustering

We performed agglomerative hierarchical clustering with Euclidean metrics and 4 different methods to compute the inter-cluster distance: `Single`, `Complete`, `Average` and `Ward`. As we expected, dendrograms produced with the `Single` method show lower distances between each bifurcation,

while those produced with `Complete` method have higher distances. The resulting dendograms are shown in Figure 14 along with the threshold where we stop merging the clusters. We also reported the size of each cluster and the Silhouette score in Table 2. Despite having one of the lowest Silhouette, the `Ward` method gives the most useful clusters, while the other three methods produce imbalanced clusters. For this reason, we will focus our analysis on the `Ward` results.



(a) Single (b) Complete (c) Average (d) Ward

Figure 14: Dendograms of hierarchical clustering with 4 inter-cluster distance methods.

Method	#Clusters	Cluster Size	Silhouette
Single	4	(1914, 2, 1, 1)	0.562
Complete	5	(1141, 682, 58, 27, 10)	0.141
Average	5	(1847, 40, 18, 12, 1)	0.404
Ward	5	(1030, 385, 302, 138, 63)	0.162

Table 2: Sizes of the clusters of hierarchical clustering.

The boxplots of some interesting attributes obtained with the Ward methods are found in Figure 16. We can notice that the clustering results are very similar on some attributes, but are overall harder to characterize with respect to k-means. We highlight some key differences as follows:

- Purple (63 players): identical profile to the *Professionals* profile obtained with k-means.
- Red (302 players): identical profile to the *Occasionals* profile obtained with k-means.
- Orange (1030 players): the most common cluster captured by the Ward method; unfortunately it has no immediate characterization, since essentially all attributes correspond with the global average.
- Blue (138 players): a very similar profile to the one captured in the *Standard player* profile, with distinctively high `avg_1stWin%`, `avg_ace%`, `avg_svptWon%`, and `avg_bpSaved%`, and a preference for the `hard%` surface. However, the high difference in size of the clustering between the two methods might indicate that Ward failed to fully capture all the players profile.
- Green (385 players): has similar characteristics as the *Return hitters*, but with a higher `avg_1st%` and comparatively lower double faults `avg_df%` with respect to *Occasionals* profile. However, they still remain the non-professional group with highest `win_rate` and tournament participation.

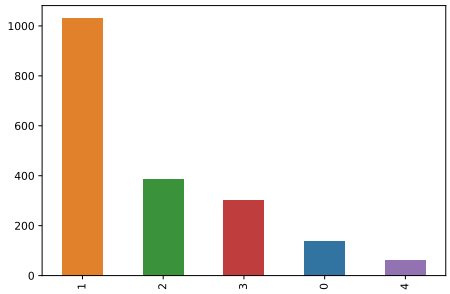


Figure 15: Sizes of the clusters for hierarchical clustering with Ward method.

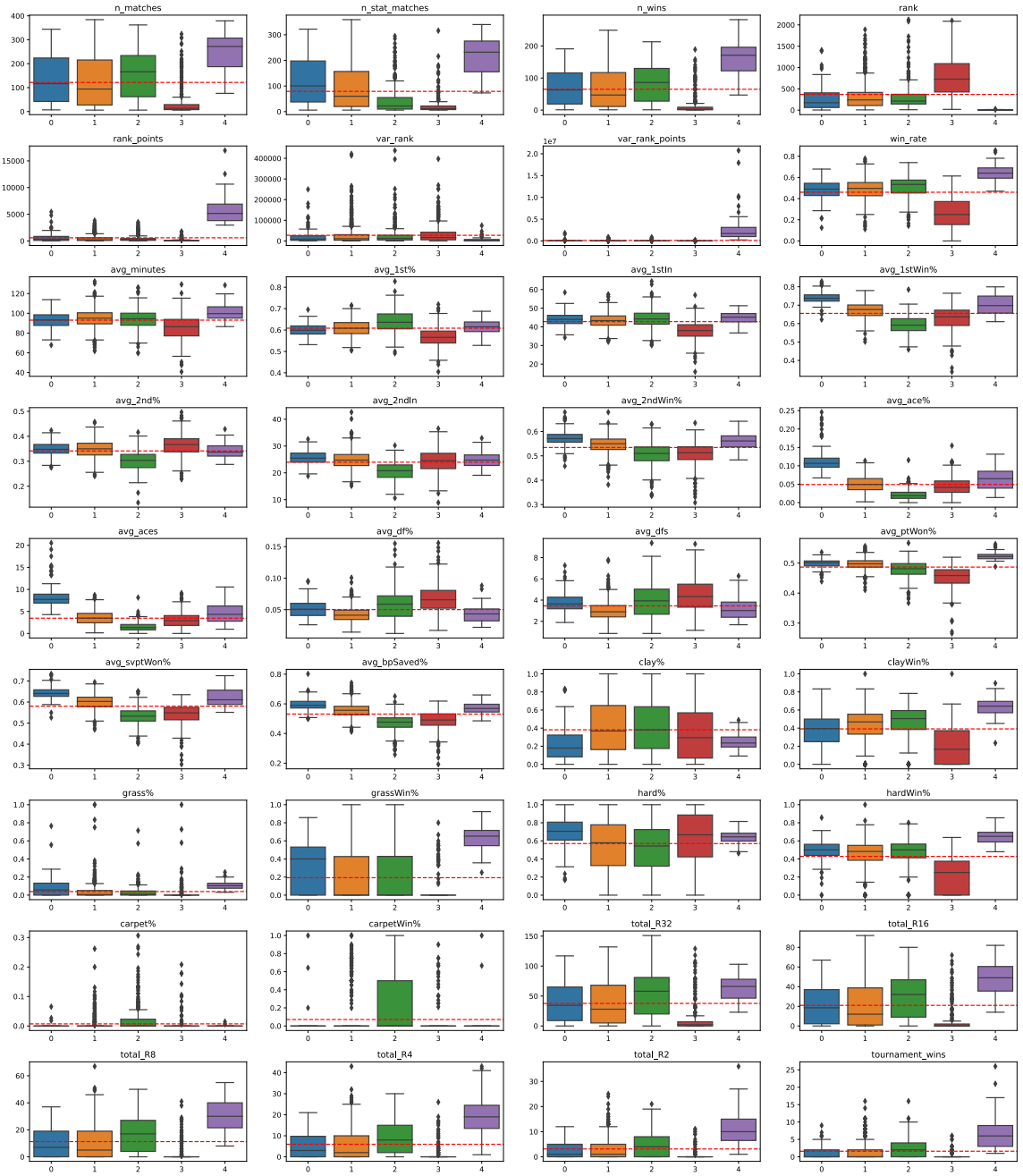


Figure 16: Boxplots of hierarchical clustering with Ward method.

3.5 Optional: Fuzzy C-Means

This clustering algorithm works similarly as k-means. However, in Fuzzy C-Means, a data point can belong to multiple clusters, i.e. it has a degree of belonging to a cluster. The parameters of this algorithm are the number of clusters k and the degree of the cluster's fuzziness m, which are determined by performing grid search in range [3, 10] for k and [1.5, 4] for m by maximizing the Silhouette score and minimizing the Davies-Bouldin score. The centroids have been initialized with K-Means++.

The three best parameter combinations we found are reported in Table 3. In the case of $k = 3$, the players are partitioned based on their skills: good (4%), bad (33%), and average (62%), and they are easily distinguishable by looking at the win rate, rank, and rank points. Other statistics such as `avg_ace%` and `avg_df%` also correspond to this characterization. However, despite the higher silhouette score with respect to the other cases, the granularity of the partitioning is still too coarse and does not characterize them on different play styles.

k	m	**Silhouette**	**Davies-Bouldin**	**Cluster sizes**
3	1.1	0.21	1.55	(1199, 630, 89)
5	1.5	0.19	1.37	(641, 531, 364, 306, 76)
6	1.1	0.20	1.31	(684, 521, 336, 269, 89, 19)

Table 3: Best parameter combinations of FCM.

In the more interesting case with number of clusters $k = 5$, on the other hand, the characterization is very similar to the one of k-means, which is confirmed by their very similar Silhouette score, 0.199 of FCM compared to 0.203 of k-means, along with the similar cluster sizes.

3.6 Optional: BSAS

The Basic Sequential Algorithmic Scheme (BSAS) is a simple linear-time clustering algorithm which depends on only two parameters: the maximum number of clusters m and the threshold t. The latter parameter indicates the maximum distance between the centroid of a cluster and its elements. BSAS works by initializing the first cluster with a single data point, then searches for the nearest point to it and if their distance is less than t it adds it to the cluster and updates the centroid. Meanwhile, if the distance is larger than t, it initializes a new cluster with the data point. The procedure then iterates over all the data points. In our analysis, we tried to vary the maximum number of clusters m from 2 to 10 but settled on $m = 5$ since it was at least able to distinguish the best players from the others. The 5 clusters produced have the following number of players in it: $[878, 799, 181, 52, 8]$. The only meaningful characterization is provided by the last two clusters containing in total the 60 best players.

3.7 Comparison of results

To conclude, we experimented with five clustering algorithms in this project, each with different results:

- K-means provided the best results with respect to the characterization, giving somewhat easy to distinguish clusters based on their statistics and skill level: best, worst, and average, with the average being further divided into 3 clusters with distinguishable playstyles.

- DBSCAN does not provide good results, but this is expected by looking at how the players are distributed. The average players are close together in a homogeneous group, while the best and worst ones are further apart from the average and end up being identified as noise. This is problematic when choosing the `eps` parameter since to cluster together the best players we would need an high `eps`, but this would in turn group together all the average players together. Given the distribution of the players, we expect for other density-based algorithms to perform poorly on this dataset.
- Hierarchical has the advantage of being the easiest to interpret graphically with the dendograms, but the results greatly depend on the method used to merge the clusters together, with only the `Ward` method giving satisfactory clustering results. The `Ward` method gives a partitioning somewhat similar to k-means on the best and worst players, but fails to fully characterize the average group which ends up being collapsed in a single big cluster.
- Fuzzy C-Means produces similar characterization to k-means, as can be expected by the similarly working algorithm. We believe this method could provide an equally good alternative to the k-means clustering.
- BSAS splits the best players into 2 small clusters. This is caused by the high distances between the best players, some are greater than the algorithm's threshold. We could increase this threshold to merge them, but it will also merge the clusters of average players, hence providing no meaningful characterization.

4 Classification

The aim of this task is to predict a custom binary indicator `high_profile` referring to whether a player is among the best ones. We considered using our clustering results with k-means and considering high profile players as the *Professional* profile set; we decided against this choice since the group was too small (80 over 1918 profiles, 4%) and would have been hard to identify and train for. Instead, we define `high_profile` as the top 15% percentile of the players on the composite indicator `performance`, defined as follows:

$$\texttt{performance} = \frac{\texttt{rank\_points}}{\texttt{rank}}$$

In Figure 17 we report the rank points of the two groups with respect to win rate and rank: we highlight the elbow-like partitioning with respect to the rank, with which we confirmed the threshold value.

(a) Rank points over rank

(b) Rank points over win rate

Figure 17: High/low profiles splitting over rank points.

Unfortunately, to classify data we could not reuse the attribute sets used in clustering, since they included `rank_points`. Therefore, we constructed a new classification profile that statistically describes the players' performance and we then filtered it such that once again all attributes have low pairwise correlation. This is done to both aid the models in classification, e.g.: for the case of neural networks, as well as to reduce the curse of dimensionality. We selected the following 9 attributes: `avg_ace%`, `avg_df%`, `avg_1st%`, `avg_2ndWin%`, `avg_bpSaved%`, `avg_ptWon%`, `win_rate`, `n_matches`, and `tournament_wins`.

4.1 Methodology

We analyze the performance of 7 different models: k-NN, Logistic Regression, Naive Bayes, Decision Trees, Random forests, Support Vector Machines and Neural Networks. Before training the models, we have shuffled and then physically removed 20% of the data in a separate file to use it as a test set at the end of our model construction. All input attributes used for classification were normalized using `StandardScaler`. In order to search for the hyperparameters of each model, we used the *k-fold cross-validation method*, using one third of the train data as validation ($k = 3$) and using the average model accuracy over the folds to evaluate each hyperparameter configuration.

4.2 Unbalanced data

To deal with the fact that the high profile class is underrepresented in the dataset, we compared where possible two approaches: the first is to assign different weights to the classes such that the cost of mispredicting the minority class is penalized by a larger weights; the second one is by using the SMOTE algorithm, which over-samples the minority class in order to have a balanced distribution of the target. We found SMOTE to provide more reliable results, and we present a comparison between models trained using SMOTE and for the unbalanced case in 4.4.

4.3 Models

We present here a brief description of the models and approaches we experimented with, reporting the grid search ranges and the best hyperconfigurations only for the unbalanced case.

k-NN

We perform a grid search over the following parameters and values reported in Table 4, with the results selected by the grid search reported in **bold**:

Parameter	Values	Best Unbalanced	Best Balanced (SMOTE)
`k`	$[2, 3, \ldots, 20]$	**11**	**2**
`weights`	[uniform, distance]	**uniform**	**uniform**
`algorithm`	[ball_tree, kd_tree, brute]	**ball_tree**	**ball_tree**
`norm`	[1, 2]	**1**	**2**

Table 4: Grid search performed for k-NN.

Naive Bayes

For Naive Bayes we only consider the empirical distribution of the training target as the prior distribution.

Logistic Regression

Logistic Regression seeks for the linear decision boundary that minimizes the cross entropy loss. For this model, we compare the results when trying different regularization coefficient C and different solver algorithms. For C, we uniformly take 30 values between 0.0001 and 0.5. While for the solver algorithm, we try `newton-cg`, `lbfgs`, `liblinear`, `sag`, `saga`. The best parameters for unbalanced dataset are $C = 0.059$ and the `liblinear` solver.

Decision Tree

We search over the `max_depth` from 2 to 20, the `min_samples_split` from 3 to 5, the `min_samples_leaf` from 4 to 5, and the criterion either `gini` or `entropy`. The best decision tree found uses `gini`, a maximum depth of 3, 4 minimum samples per leaf, and 3 minimum samples per split. A graphical representation of the tree can be seen in Figure 18.

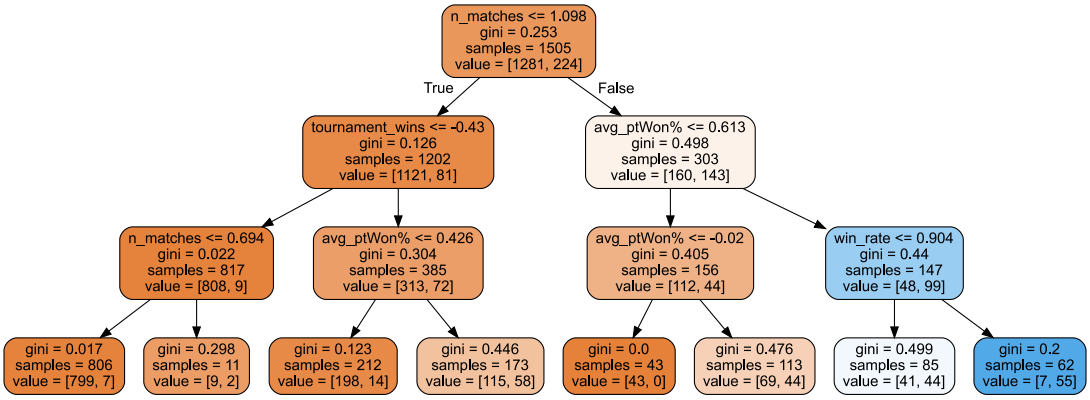


Figure 18: Decision Tree obtained with the grid search.

Random Forest

We perform a grid search over the following parameters and values reported in Table 5, with the results selected by the grid search reported in **bold**:

`forest_size`	**30**
`bootstrap`	[**true**, false]
`criterion`	[entropy, **gini**]
`max_depth`	$[2, 3, \ldots, 20]$, best=**9**
`min_samples_split`	$[1, 2, \ldots, 10]$, best=**8**
`min_samples_leaf`	$[1, 2, \ldots, 10]$, best=**7**

Table 5: Grid search performed for Random Forest.

SVM

Since SVM allows for a greater number of numerical hyperparameters, we first started with a broad grid search and then we iteratively refined the search near the chosen value, for which

we report in Table 6. We also examine different choices of kernel and obtained the best results using the `rbf` kernel, which is expected given the Gaussian distribution of many attributes in the dataset.

`C`	[0.0001 - 2], best=**1.249**
`gamma`	[0.0001 - 0.5], best=**0.439**
`shrinking`	[**True**, False]
`probability`	[**True**, False]
`kernel`	{**rbf**, sigmoid, poly}
`poly_degree`	[1,2,**3**,4,5,6]

Table 6: Grid search performed for SVM.

Neural Network

For Neural Networks, we decided against using k-fold in favor of a simple holdout technique with 30% validation set, since this allowed us to explore more models in less time. The validation set was separated from training before oversampling with SMOTE. The parameters and values analyzed in the grid search can be found in Table 7: we used the Adam learning algorithm with a batch size of 256 along with Batch Normalization to stabilize the training and reduce overfitting, and for the output layer we use the sigmoid activation function with Cross Entropy to compute the loss. Each model is trained for a maximum of 1500 epochs and the final number of epochs is chosen with an Early Stopping technique monitoring the validation loss with patience set to 300.

Parameter	Values	Best Unbalanced	Best Balanced (SMOTE)
`lr`	[1e-5 - 1e-2]	**0.01**	**0.00667**
`hidden_layers`	[2,3]	**3**	**3**
`hidden_units`	[8,16,32,64,128,256]	**128**	**128**
`activation function`	{relu, tanh}	**tanh**	**relu**
`dropout`	[0, 0.1, 0.3, 0.5]	**0.3**	**0.5**

Table 7: Grid search performed for NN.

4.4 Results

We compare the performance of the models so far presented by studying the accuracy, precision, recall and F1-score on the training set in Table 8 and test set in Table 9. In Table 10 and Table 11 we present the same model approaches but trained on a SMOTE-balanced dataset, with hyperparameters chosen on the same ranges but with a separate grid search. Given the unbalanced distribution of the two classes, we need to look at both the accuracy as well as the precision and recall of the two classes in order to make a fair comparison of the models. In Table 8 to Table 11 we use `A` to refer to Accuracy, `P`i for *Precision* on the class i, and similarly we write `R`i for *Recall* on the class i. Since comparing pairs of recall and precision scores can be difficult, we also report the F_1-score for both classes as an additional metric to aid in model comparison.

Train							
Model	A	P0	P1	R0	R1	F_1:0	F_1:1
k-NN	0.91	0.93	0.79	0.97	0.57	0.95	0.66
Logistic regression	0.88	0.90	0.68	0.97	0.41	0.93	0.51
Naive Bayes	0.84	0.95	0.47	0.85	0.75	0.90	0.58
Decision tree	0.89	0.91	0.67	0.96	0.44	0.93	0.53
Random forest	0.94	0.94	0.92	0.99	0.67	0.97	0.78
SVM	0.93	0.93	0.90	0.99	0.59	0.96	0.72
NN	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 8: Result on the train dataset.

Test							
Model	A	P0	P1	R0	R1	F_1:0	F_1:1
k-NN	0.90	0.93	0.71	0.95	0.63	0.94	0.67
Logistic regression	0.88	0.90	0.67	0.96	0.44	0.93	0.53
Naive Bayes	0.86	0.96	0.53	0.86	0.83	0.91	0.64
Decision tree	0.88	0.91	0.69	0.96	0.46	0.93	0.55
Random forest	0.90	0.92	0.72	0.96	0.56	0.94	0.63
SVM	0.90	0.92	0.78	0.97	0.54	0.94	0.64
NN	0.89	0.92	0.69	0.95	0.58	0.94	0.63

Table 9: Result on the test dataset.

Train (SMOTE-balanced data)							
Model	A	P0	P1	R0	R1	F_1:0	F_1:1
k-NN	0.99	0.99	1.00	1.00	0.99	1.00	1.00
Logistic regression	0.85	0.88	0.82	0.80	0.89	0.84	0.85
Naive Bayes	0.85	0.90	0.82	0.80	0.91	0.84	0.86
Decision tree	0.93	0.95	0.91	0.91	0.96	0.93	0.93
Random forest	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SVM	0.96	0.99	0.92	0.92	0.99	0.95	0.96
NN	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 10: Result on the train dataset, with oversampling.

Test (SMOTE-balanced data)							
Model	A	P0	P1	R0	R1	F_1:0	F_1:1
k-NN	0.87	0.94	0.56	0.90	0.68	0.92	0.62
Logistic regression	0.86	0.99	0.53	0.85	0.93	0.91	0.68
Naive Bayes	0.83	0.98	0.48	0.81	0.93	0.89	0.63
Decision tree	0.87	0.97	0.56	0.88	0.83	0.92	0.67
Random forest	0.90	0.96	0.64	0.92	0.80	0.94	0.71
SVM	0.88	0.96	0.59	0.90	0.80	0.93	0.68
NN	0.91	0.95	0.70	0.94	0.71	0.94	0.71

Table 11: Result on the test dataset, with oversampling.

4.5 Model comparison

For the unbalanced case, we can notice that all models reach around 90% accuracy, which is expected given the 85%/15% distribution of the two classes. *k-NN*, *Random forest*, and *SVM* models seem to have the best performance on the test set, with an average F_1 score on the positive `high_profile` class of 65%. In particular, k-NN gives the best compromise of both accuracy and F_1 score. In terms of accuracy, the worst classifier is *Naive Bayes*, which is not surprising given the absence of tuning and low configurability of the model. While all other models have good Precision and low Recall on the positive `high_profile` class, *Naive Bayes* provides a different compromise with low P1 and high R1, while still achieving a relatively good F_1:1 score of 0.64. *Neural Networks* provided satisfactory results despite the unbalanced distribution of the training data, but we predicted for this situation to improve by training on the balanced dataset.

For the balanced case, we notice that SMOTE improved the F_1 score while decreasing the accuracy for almost all models. *Neural Networks*, on the other hand, improved both F_1 scores and accuracy. The model chosen by the grid search is also less complex than the unbalanced case, given the higher dropout and `relu` activation function, suggesting the balanced task to be easier for NN. Random forest also maintains the same accuracy. Another exception is k-NN, where using SMOTE decreased both accuracy as well as the overall performance: we somewhat expected these results, since k-NN is sensitive to training data and especially to the synthetic patterns generated by SMOTE, which introduce an artificial bias for the underrepresented class on the neighbours computation.

Even though Neural Networks gives the highest accuracy, Random Forest also provides a good compromise between performance and interpretability of the model. We believe that classification with the other models might improve by considering more complex balancing methods other than SMOTE that better capture the underlying distribution.

5 Time Series Analysis

The dataset provided for this task is the monthly temperature of 100 cities for 10 years. All attributes of the 12000 rows in the dataset are correct and complete, so there was no need for data cleaning to be performed. Our goal is to find groups of similar cities with respect to the temperature trends. To do this, we first transform the dataset into a 2-dimensional array of size $(100, 120)$, where the rows correspond to a city and the columns correspond to a month of a year. In 5.1 and 5.2 we describe how the groups of similar cities were discovered by considering the whole time series of temperatures using both k-means and hierarchical clustering. Finally, we describe in 5.3 and 5.4 other approaches by representing the time series with a set of global features and by approximating the time series in a smaller dimension.

5.1 K-means

We considered 2 metrics to compute the similarity between time series: Euclidean and Dynamic Time Warping (DTW). The number of clusters k is obtained using the Elbow method by looking at the SSE, Silhouette, and Davies-Bouldin score. We decided to pick $k = 6$ for Euclidean and $k = 5$ for DTW, for which we report the results in Table 12.

Method	#Clusters	Cluster Size	Silhouette	SSE
Euclidean	6	(33, 29, 14, 14, 9, 1)	0.431	1012.44
DTW	5	(30, 26, 19, 15, 10)	0.388	483.00

Table 12: Sizes of the clusters with k-means and different similarity metrics.

To evaluate the quality of our clustering results, we compute the average temperature of each city and plot the values in increasing order. However, this does not show how the temperature trends and changes over the year, so we also plot the average temperature of each cluster in all of the 120 months, which is shown in Figure 19. We can see that both Euclidean and DTW produce clusters with distinguishable temperature trends. An interesting difference is that the additional cluster given by Euclidean (cluster black) contains only 1 sample for the city of Santiago (Chile). In DTW, this singleton is merged with cluster blue as it has the most similar trend shape, despite their misalignment in the time series. Euclidean metric can not capture this similarity because it aligns each observation one on one linearly.

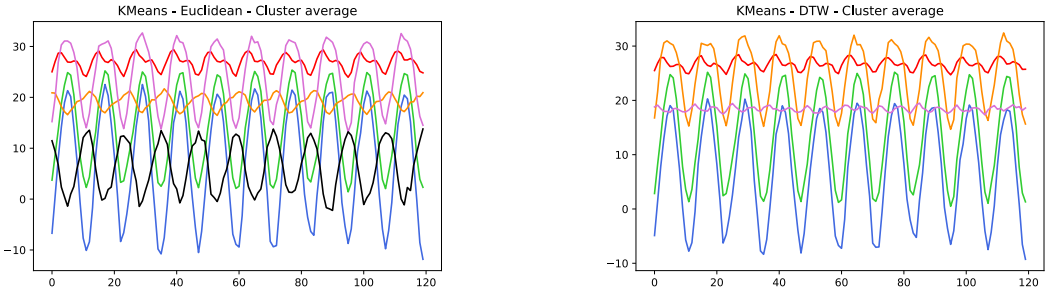


Figure 19: Average temperature over months for each cluster.

In addition, we also plot the cluster distribution in the world map, which is shown in Figure 20. This helps to understand that cities along the same latitude are grouped together. We only report the world map for Euclidean metric since the result is qualitatively similar to DTW.

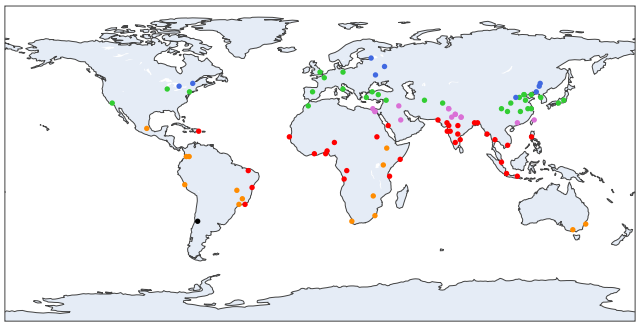


Figure 20: Clusters distribution in the world map (clustered using k-means with Euclidean).

5.2 Hierarchical clustering

We performed agglomerative hierarchical clustering with 4 methods to compute the inter-cluster distance. For the similarity metric, we only used Euclidean since the library does not support DTW. The results are reported in Table 13.

Method	#Clusters	Cluster Size	Silhouette
Single	4	(61, 36, 2, 1)	0.411
Complete	6	(38, 29, 14, 9, 9, 1)	0.399
Average	5	(47, 29, 14, 9, 1)	0.440
Ward	6	(33, 24, 15, 14, 8, 6)	0.387

Table 13: Sizes of the clusters of hierarchical clustering.

By considering the similar Silhouette score and the world map, all 4 methods produce similar groupings again based on latitude. They also present the same singleton cluster as the one in k-means, with the Ward method again being an exception and performing comparably better than other methods with better cluster size distributions.

5.3 Feature-based clustering

We then experimented with feature-based clustering by extracting 13 statistical features from each time series, e.g mean, median, standard deviation, etc., and then we cluster the cities based on these 13 attributes using k-means. We set $k = 5$ to be able to compare these results with the ones obtained by considering the whole time series. The resulting Silhouette score is much lower (0.20 compared to 0.38-0.44). Plotting the average temperature over time of each cluster and visualizing them in the world map also did not show any interesting patterns.

5.4 Other clustering approaches

Another approach we applied is to reduce the time series from containing 120 observations into 12 observations. This is done by taking the average temperature of a specific month in 10 years. This increases the Silhouette score to 0.29, but again we unfortunately found no significant patterns in the groupings.

6 Conclusion

We explored several clustering and classification algorithms on a tennis matches dataset. While figuring suitable models and tuning the parameters is important, preparing and cleaning the data also played a crucial part in the data mining process. We started by understanding the dataset through visualization and small experiments, then proceeded by improving its quality and determining the most relevant features. We found this and the cluster intepretation phase to be particularly challenging due to our limited knowledge of the domain, which we progressively learned along with the dataset.

From the clustering analysis, we found interesting patterns on the dataset which not only group the tennis players based on their skills, but also their playing style. Since the dataset unfortunately contained a great percentage of matches with null statistics, these results probably could be improved if the data contained complete players statistics for each match, or included statistics on the return games. Other patterns may also be revealed in the clustering by considering male and female players separately, in case both genders have different criteria to be considered as good players. We found that not all clustering algorithms worked well for this dataset. For example, we believe for density-based algorithms to be not suitable on this dataset because the average players are densely packed together while the best players are sparsely distributed and hard to group by density.

For the predictive analysis, the models we tested were able to predict the class of tennis players with satisfactory results, especially the case of Neural Networks and Random forests possibly due to the larger and more flexible model. Balancing the dataset using SMOTE decreases the accuracy of most of the models, but it also increases the F1 score of the high profile label which we believe gives a more reliable result given the unbalanced target used for classification.