1. Data Understanding and Preparation

1.1: Description of the *Tennis Matches* dataset

The dataset contains information about some tennis matches held during the period from 2016 to 2021. It is composed by specifying some personal and technical details about:
- *tourneys* such as the *Name*, *Spectators, Revenues* and *Levels;*
- *matches* such as the *Tourney* they belong to, the type of *Surface*, the *Date* and the *Scores*;
- *players* such as *Names, Genders, Heights, used Hands, Nationalities, Entries* and served *Services;*

By just summarizing and including in some domains these attributes is introduced some semantic meaning. But each of them will be described, step by step, in a more detailed way, as follows.

At the beginning, the dataset is splitted in several csv files that are put all in one since, in this way, it is possible to have a global view. This preliminary step could be named ***data integration*** and refers to the joining of the Tennis Matches table with the one of the Players name table. Since this last is divided into two further csv files, one for the male players and one for the female players, this information about the gender continues to be preserved in the joining part. The players not matched in these two files and so with no specific gender, are marked with label ‘U’ that stays for unknown gender, while male and female are identified with label ‘M’ and ‘F’.

There are in total 51 attributes with 186128 entries, initially it is decided to drop duplicates by reducing the number of records to 185819, so by quite 300 elements. It is noted that there are a lot of null values for each attribute which are treated deeply later on.

1.1.1 The Semantics, Statistics and Distribution of the *Tennis Matches* dataset

The used libraries in this part are: pandas, numpy, scikit-learn, matplotlib.
Analyze each attribute by shortly describing them with also the type, the amount of missing values. Visualize some graphs.

Tourney

- **tourney_id: object, [NaN values = 55]**

  Has to be unique and the first 4 characters are always the year while the remaining part is random.

  Is split the value of it in 2 parts, the first with 4 characters that represent tourney year and the second with the remaining characters. Between the year parts are discovered 6 several years in which the matches have been played that are from 2016 to 2021.
  The only check that is possible to do is about the first 4 characters that represent tourney year, if it's not correct then the corresponding row is eliminated. In this case, no year of tourney_id is wrong, so no row is dropped.

- **tourney_name: object, [NaN values = 25]**

  Is the name of the tourney.

  The unique names turn out to be 2488. Each name is composed either by the name of the places where the matches have been played, or the nationality of the competition.

- **tourney_spectators: float64 → int64 [NaN values = 27]**
- **tourney_revenue: float64 [NaN values = 26]**

Total amount of spectators and revenue in a tournament.

These attributes are always studied together since, as seen in the next graph, they have a linear behavior, so i.e. there is a greater revenue if there is a greater amount of spectators.

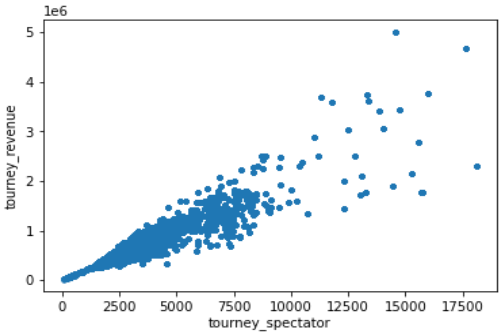

Fig.1.1.1.1 Plot between spectators and revenues

- **tourney_level: object [NaN values = 29]**

Some are to be different between men and women and some are instead shared without distinction of the gender.

There are 15 unique values.

In the right table are shown in blue the values for male players, in pink those for the female ones and in black those for both the genders.

In the graph below, is shown the distribution of the level's attributes according to the gender of the players. It has to be noted that the database contains more matches between females with respect to matches between males while matches with at least one unknown gender are very few. Note: this result does not imply that females are more than males because this distribution represents the number of matches with male or females and not the cardinality of gender players.

Fig.1.1.1.2 Gender, Tourney Levels, description

M	**G**	Grand Slams
M	**M**	Masters 1000s
M	**A**	other tour-level events
M	**C**	Challengers
M	**S**	Satellites/IFTs
M	**F**	Tour finals and other season-ending events
W	**P**	Premier
W	**PM**	Premier Mandatory
W	**I**	International
W	**i.e. 15 = ITF $15,000**	various levels of ITFs given by the prize of money (in thousands)
W	**i.e. T1 = Tier I**	older WTA tournament designations
M/W	**D**	W: Federation/Fed/Billie Jean King Cup, Wightman Cup and Bonne Bell Cup M: Davis-Cup
M/W	**E**	exhibition (events not sanctioned by the tour, though the definitions can be ambiguous)
M/W	**J**	juniors
M/W	**T**	team tennis

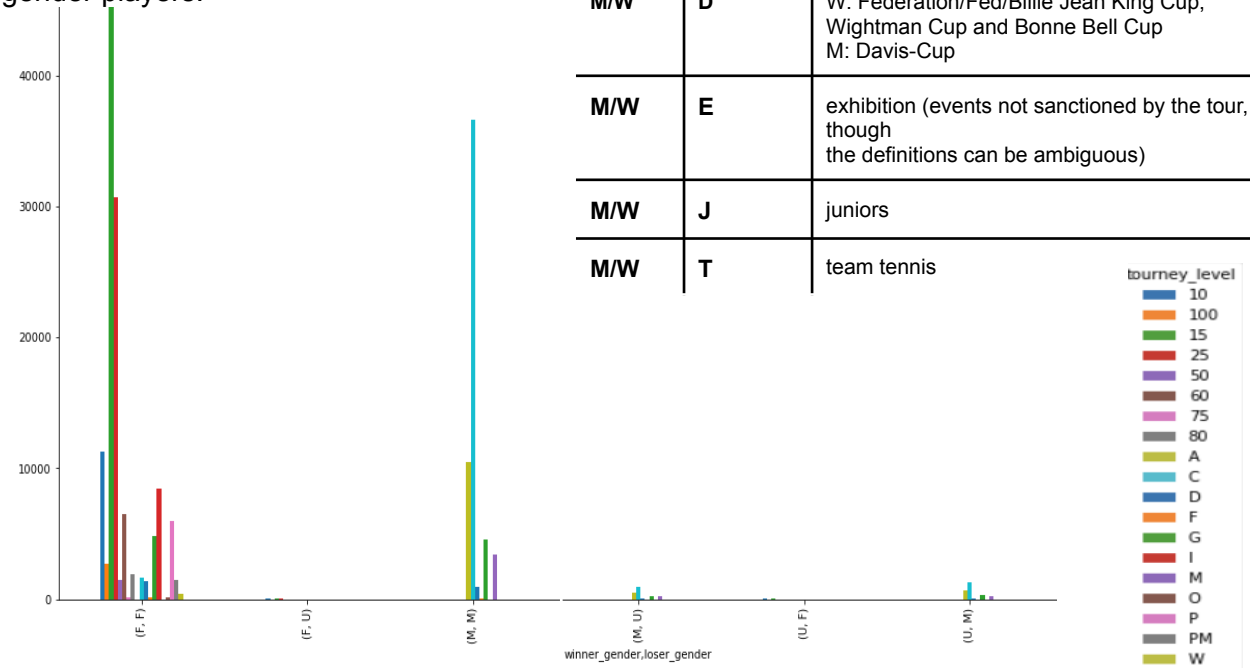

Fig.1.1.1.3 Gender, Tourney Levels, distribution

Matches

- **match_num: float64 → int64 [NaN values = 27]**

  It is a match specific identifier; often from 1 to 300 and other times is arbitrarily chosen.

- **surface: object [NaN values = 188]**

  It specifies the kind of surface of the match.

  As it can be seen graphically, there are 4 different types of surfaces among the years. It usually follows the same increasing trend of distribution except for the last year in which there is an inversion of the 'Clay' with the 'Hard' value. Is noted also that the number of played matches is fewer in the last two years than the previous ones because, in minimum part for the null values and mainly because there weren't effective matches, this fact is for the pandemic situation that has deleted entire tournaments.

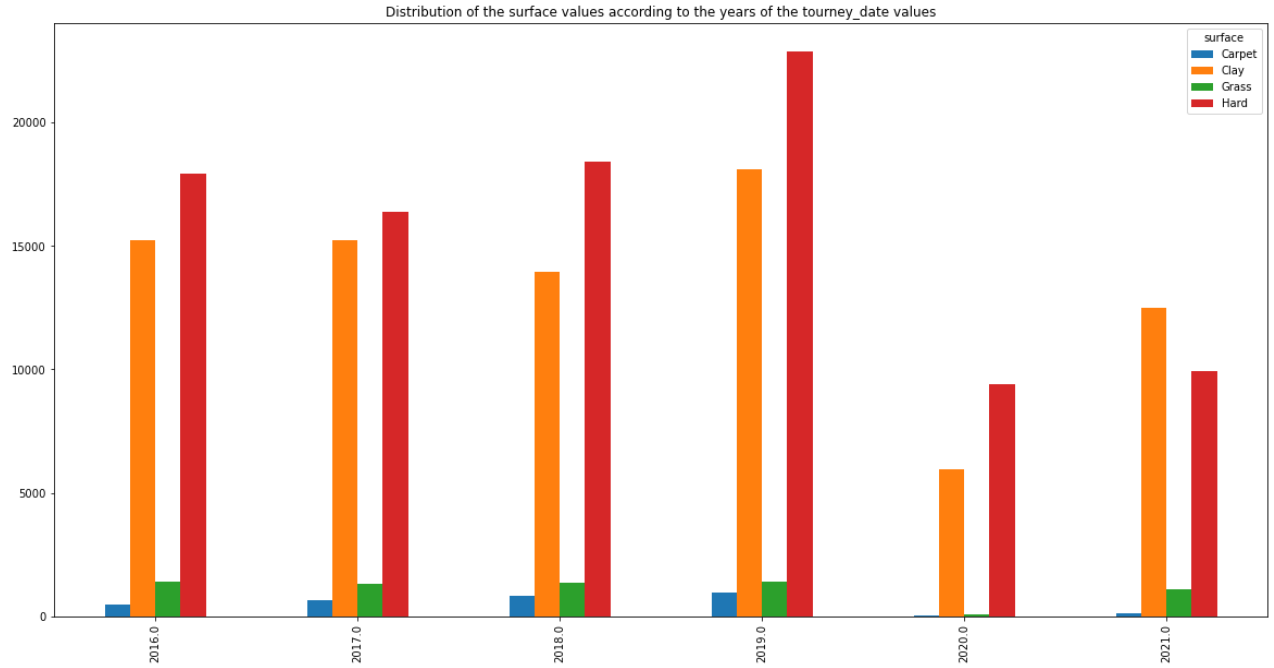

Fig.1.1.1.4 Distribution of the surface according to the years of tourney_date values

- **score - object [NaN values = 193]**

  Represents the score of each match.

  There are many unique values between the various games.

- **round - object [NaN values = 30]**

  Is a series of matches in an elimination tournament, the winners of which advance to the next round.

  The 12 unique values are: F, SF, QF, R16, R32, R64, R128, Q1, Q2, Q3, RR, BR

- **best_of - float64 [NaN values = 29]**

  '3' or '5', indicating the number of sets for each match.

  The values are always of the 2 specified types, the most common is 3, as confirmed in the graph on the right.

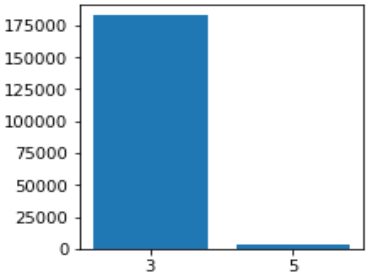

Fig.1.1.1.5 Histogram distribution of best_of

- **draw_size: float64 → int64 [NaN values = 29]**

  Number of players in the draw, often rounded up to the nearest greater power of 2;
  i.e. a tournament with 28 players may be shown as 32, if of 15 then as 16, if of 23 then as 32 and so on.

  Since all the powers of 2 are integers the idea is to convert to the 'int' format. Then use a logarithmic function to take the exponent to be used to get the wanted power of 2.
  After doing this, the following 7 unique values are obtained: 2, 4, 8, 16, 32, 64, 128.

- **tourney_date: float64 → Datetime64 [NaN values = 28]**

  It specifies the date of the match which usually consists of the Monday of the tournament week.
  It is composed of 8 digits in the YYYYMMDD format.

  Firstly, it is converted to the right type.
  This attribute is used mainly for statistical purposes, as shown in the following graph the distribution of the played matches among the months of the years. Is noted that: the months with the most played matches are July and September in 2019, while before they are July and October. Contrarily, in each distribution, December is the one with the fewest played matches. These considerations are not valid for the last 2 years, always for the Covid pandemic disease.

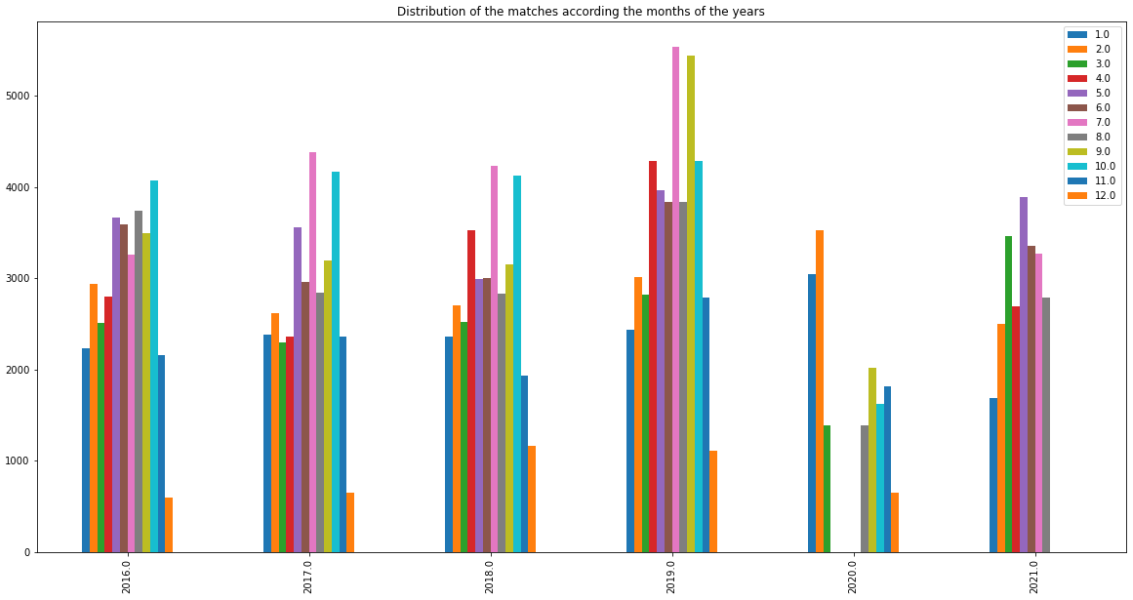

Fig.1.1.1.7 Matches' months distribution among the years

- **minutes - float64 [NaN values = 104461]**

  Represents the match length in time, where available.

  The minutes are kept of type float to facilitate the statistical control of the data.
  By looking at the description it is noted that the min=0 and max=4756=quite 80 hours, that are for sure outliers.

  To visualize its graphical distribution the attribute has been transformed in hours. Since it is assumed that the majority of played matches don't last more than 11 hours then from this threshold and over they are all considered outliers.

  There are a lot of NaN values, quite the half of all the rows. Both the great number of outliers and NaN values will be deeply treated in the later steps.

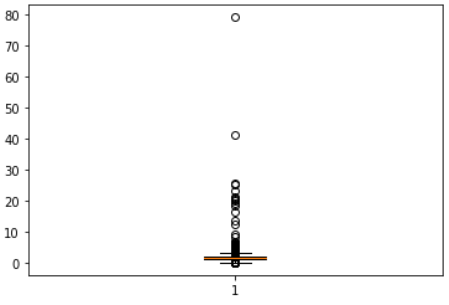

Fig.1.1.1.8 Matches' hours distribution

Players

- **winner_id: float64, [NaN values = 55] | loser_id: float64, [NaN values = 28]**

  Identify the player, both if the winner or loser, of the match.

  They have been treated to verify if there is an unique correspondence among each player's id with the name it refers to, and vice versa, among each player's name with the id it refers to. Since more players have the same associated id and more ids belong to several different names then they cannot be used to check uniformly someone, they have to be analysed carefully together with other attributes when referring to a specific player.

- **winner_name: object [NaN values = 27] | loser_name: object, [NaN values = 31]**

  Name of the player.

  It is seen that the count of unique values among the winners is less than that of the losers, this means that the winners are a small subset of all the players.

- **winner_ioc: object, [NaN values = 29] | loser_ioc: object [NaN values = 26]**

  It is a 3 character country code

  Each player has its own country of reference, usually it is the native country but sometimes it is also another country a player can represent too.

- **winner_hand: object [NaN values = 46] | loser_hand: object, [NaN values = 98]**

  R=Right, L=Left, U=Unknown. It is the player's hand.

  The graph on the right shows the distribution of players' hands among the matches. Quite the half of the matches played is between, both winner and loser players, with the R hand. The present number of U hands are greater than the ones of the L hand.

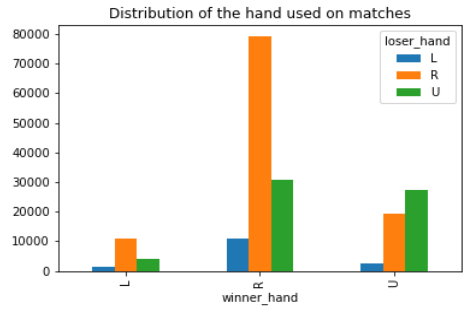

Fig.1.1.1.9 Distribution of the hand used on matches

- **winner_gender: object, [NaN values = 2704] | loser_gender: object, [NaN values = 2129]**

  M=Male, F=Female, U=Unknown. It is the gender of the player. It has been introduced during the data integration part.

  The graph on the right shows the distribution of players' genders among the matches. The ⅔ of all the dataset represents female tournaments. With reference to the U values there are very few where some are known winners males and some others are known losers males. The correspondent U part may be both F or M, it is supposed that there are not mixed matches so that the correspondent value is implicitly deduced but it could be better if contemporary checked other useful attributes such as the *tourney_level*, that is for instance dependent on the gender type.
  What is sure is that in the female tournaments there are no matches in which only one part is known and the other is not.

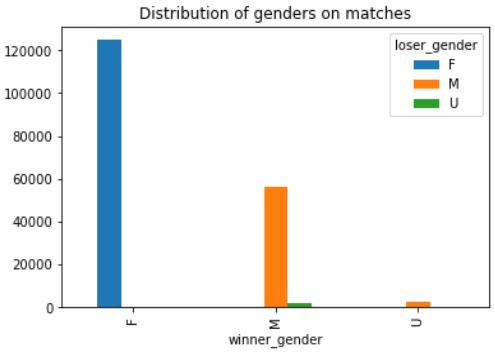

Fig.1.1.1.10 Distribution of genders on matches

- **winner_ht: float64, [NaN values = 136516] | loser_ht: float64, [NaN values = 147489]**

  Height in centimetres.

  Because a player grows during years, it could increase among the years the player has taken part, the min=2 that is an outlier and there could be others too.

- **winner_age: float64 → int, [NaN values = 2853] | loser_age: float64 → int, [NaN values = 6537]**

  Age of the player, in years, depending on the date of the tournament.

  The decimal part is not taken into consideration since it is transformed into the integer format.
  Also this attribute is something that changes during the years the player has taken part.
  If there are some non-NaN and some NaN values referring to the same player then it is easily found the missing value to replace with taking into account also the *tourney_date* attribute of which the participant has played to.
  Since max=95 then there are outliers.

- **winner_entry: object, [NaN values = 160008] | loser_entry: object, [NaN values = 141731]**

  WC = wild card, Q = qualifier, LL = lucky loser, PR = protected ranking, ITF = ITF entry, ... a few others that are occasionally used.

  There are 18 unique values, so additionally to that already introduced there are: A, SE, ALT, SR, JE, 3, 4, 6, I, P, J, IR, JR

- **winner_rank: float64, [NaN values = 19402] | loser_rank: float64, [NaN values = 35259] | winner_rank_points: float64, [NaN values = 19420] | loser_rank_points: float64, [NaN values = 35276]**

  rank: It is the player's ATP or WTA rank referring either to the *tourney_date* or the most recent ranking date before the *tourney_date*.
  rank_points: Is the number of ranking points of the player, where available.

  rank: It is difficult to deduce since it depends on the date, the only way is to have a look of how it changes throughout the various tournaments.
  An attempt to understand better their meaning is to visualize graphically their correlation. Lower is the *rank*'s value and so players with best performances, then higher is the *rank_points* attribute, and vice versa, while higher is the rank's value and so players with low performances, then lower is the *rank_points* attribute. The distribution has an iperbole behavior.

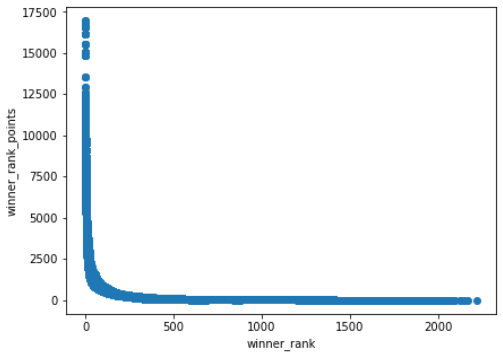

Fig 1.1.1.11: Distribution of winner_rank_points with winner_rank

The following Table represents the remaining attributes. From this simple analysis there are a lot of NaN values and these attributes are around the same quantity of missing values. Also, since, in some attributes, the avg value is distant from the max then there could be outliers. They are treated more deeply in the Cleaning phase (1.1.3) where some NaN are resolved and the possible presence of outliers are studied.

Attribute	Description	Min	Max	avg	NaN
w_ace: float64 → int **l_ace: float64 → int**	Number of aces of each player. Are transformed into integers.	0 0	75 67	4.82 3.53	**103811** **103808**
w_df: float64 → int **l_df: float64 → int**	Number of double faults of each player. Are transformed into integers.	0 0	114 114	2.85 3.16	**103809** **103802**
w_svpt: float64 **l_svpt: float64**	Number of winner serve points Number of loser serve points	0 0	1957 1672	71.29 73.57	**103809** **103810**
w_1stIn: float64 **l_1stIn: float64**	Number of first serves made	0 0	1330.0 893.0	44.27 44.56	**103811** **103817**
w_1stWon: float64 **l_1stWon: float64**	Number of first serve points won	0 0	836.0 532.0	32.14 28.04	**103809** **103810**
w_2stWon: float64 **l_2stWon: float64**	Number of second serve points won	0 0	304.0 399.0	14.46 12.71	**103812** **103809**
w_SvGms: float64 **l_SvGms: float64**	Number of serve games	0 0	49.0 50.0	11.11 10.94	**103810** **103803**
w_bpSaved: float64 **l_bpSaved:: float64**	Number of breakpoints saved	0 0	209 120	3.53 4.65	**103806** **103810**
w_bpFaced: float64 **l_bpFaced: float64**	Number of breakpoints faced	0 0	266 190	5.40 8.86	**103809** **103815**

Tab.1.1.1: Description of the last attributes

1.1.2 Correlation of the *Tennis Matches* dataset

The following correlation matrix is related to the numerical attributes of the dataset. It has to be analyzed taking into consideration the cell's colors which represent intensity of correlation between two pairs of attributes; lighter is the color then stronger is their correlation and vice versa, darker is the color then weaker is the relation. The attention is to be put on those attributes that match to lighter cells since they behave in the same manner and so do not bring further information. Since, for this are considered redundant elements then can be avoided through substituting one of them.
Among the most correlated attributes, those with correlation greater than 0.9, there are those strictly referring to the service valuation as *w_svpt* with w*_1stIn* and *w_1stWon*, w*_bpSaved* with w*_bpFaced* (the same consideration is to be thought for the losers' counterpart), and *w_SvGms* with *l_SvGms.*
Also for the *draw_size* and *tourney_specators* there is a strong relation, this suggests that higher is the table size among the competitors than higher is the number of spectators attending the matches.
Moreover, as previously introduced, and here confirmed, there is a strong relation between *tourney_revenue* and *tourney_spectators*. Some of their distributions can be viewed in Fig. 1.1.2.2. ; the first three graphs follow a linear trend together with some more distant points that represent outliers, the last one instead has a strange trend in brackets because, so there are some predefined intervals of correspondent values between the *draw_size* and the admitted *toruney_spectators*.

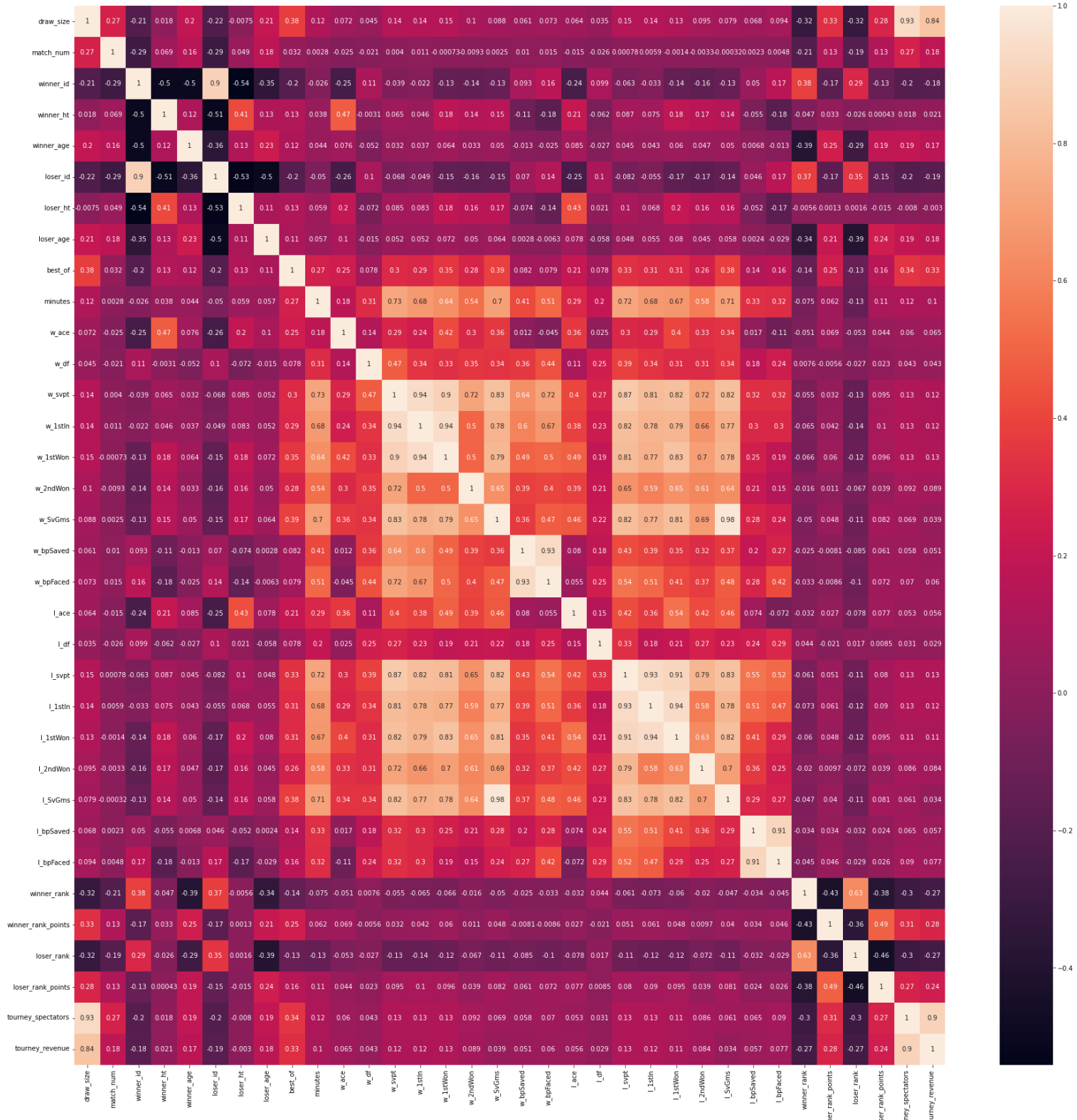

Fig.1.1.2.1: Correlation matrix for all attributes

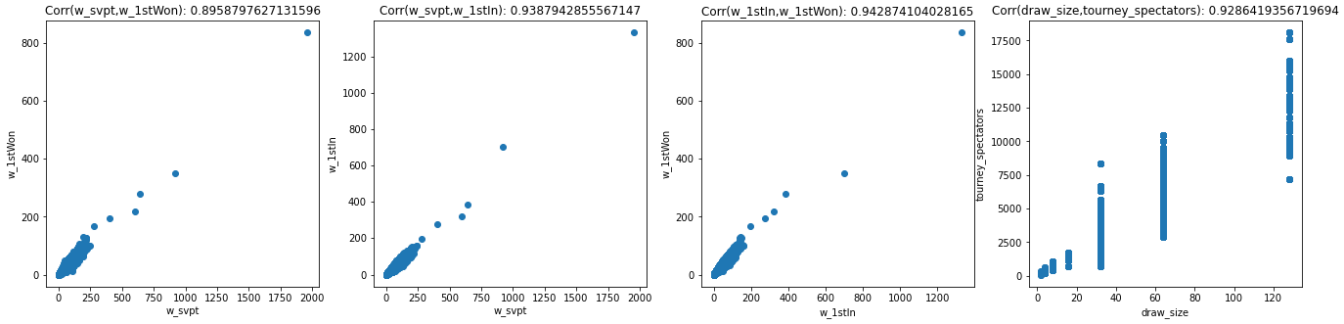

Fig.1.1.2.2: Correlation for some highly correlated attributes

1.1.3 Cleaning of the *Tennis Matches* dataset

Missing Values

The first interesting analysis is if there are rows that are duplicates of others and that have not been deleted in the previous *drop_duplicate()* function. These rows, called for simplicity twin rows, have more NaN values of their copy while the non-NaN values are in common. For this reason, it is necessary to delete the rows that result less complete than their twin counterparts since they can be replaced. To discover them is adopted a systematic approach to check which combination of NaN attribute values could achieve few rows, with the aim of reducing the complexity. After doing this, some attributes have no more NaN values at all, while attributes on the table 1.1.1 achieved a quite similar amount, that are attributes related to statistics of matches. These last attributes have quite 100'000 missing values, and so more than half of the entire dataset's rows. All the NaN values of them belong to the same rows. For them it is not possible to exploit the small amount of known values to deduce their values too.

The second consideration is the analysis of the correlation matrix, by identifying the attributes with a high point of correlation i.e. those higher than 0.90 such as ***w_SvGms*** with ***l_SvGms***, ***_svpt*** with ***_1stIn*** and ***_1stWon***, ***_bpSaved*** with ***_bpFaced***. Among them, try both polynomial and linear interpolation, obtaining usually good results for the latter, so choose the one that has the closest standard deviation to the original data. By doing this is wrong since so many missing values are among all the considered attributes so that the interpolation deduces nothing. By observing that the sum of the score values corresponds to the sum of ***w_SvGms*** with ***l_SvGms*** then add another ***SvGms*** column that replaces the homonyms ones, so delete ***w_SvGms*** with ***l_SvGms*** from the original database. Now apply the interpolation as ***SvGms*** with ***_svpt*** and ***_1stIn*** to resolve missing values of these last two attributes.

For attributes, such as ***_ace, _df, _1stWon, _2ndWon, _bpSaved, _bpFaced*** which are personal attributes, so more strongly related to the player name, consider the mean and the median methods. Also between them, choose the one with the closest standard deviation to the original data.

For ***_entry*** since they are much more than half of all the dataset rows and since they cannot be deduced from the other parts of the dataset, an idea is to remove both the 2 columns. But to decide this is too early in this phase of analysis.

With what concerns the player's ***hand****,* analyze all the various previous games associated with the same player in order to replace the missing values, where possible. Unfortunately, the search is not successful for all the NaN values, so for uniformity of the data, replace them with the U value. This absence of deducible values may be for outliers such as players that don't do so many competitions.

With reference to the ***minutes*** attribute, even if the difference between the mean and median methods is checked, apply the amount of 60.00 minutes for the unknown values because it is the one with the better similar standard deviation.

Also for the ***surface*** attribute firstly is reasoned by considering its distribution representation in Fig. 1.1.1.4, but choosing to change the missing values respecting it is avoided since it could be done bringing errors. Since these values are very few they can be easily checked and discovered something more; they refer to matches for *Fed-Cup* and *Davis-Cup*, which are national tournaments for both female and male genders. Those matches are held in different places where there is not a guarantee of the effective place and so the correspondent surface value, i.e. such as in Wimbledon where the surface is with only the *grass* value. This information can only be checked using external information manually, even if it is known the right place, then in each case they are not resolved and is continued to keep them empty.

For the ***gender*** attribute consult its distribution graph in relation with the levels of the players in Fig 1.1.1.3. Thanks to it is decided that there are only matches between players of the same gender. So, using this fact, find the gender of the missing player, such as: if the gender of the winner is known then it is known also the gender of the loser, and vice versa. Consulting another time the updated graph, there is a helpful distribution of the colors that suggests the *tourney_levels* of each of them, they all seem to be of M gender. To be sure about this fact are printed the *tourney_levels* of the players with U gender and are effectively of the M gender, so can be replaced in the right way. This is a trustable graph since there are no errors among the distribution of the colors identifying the *tourney_levels* with the right gender.

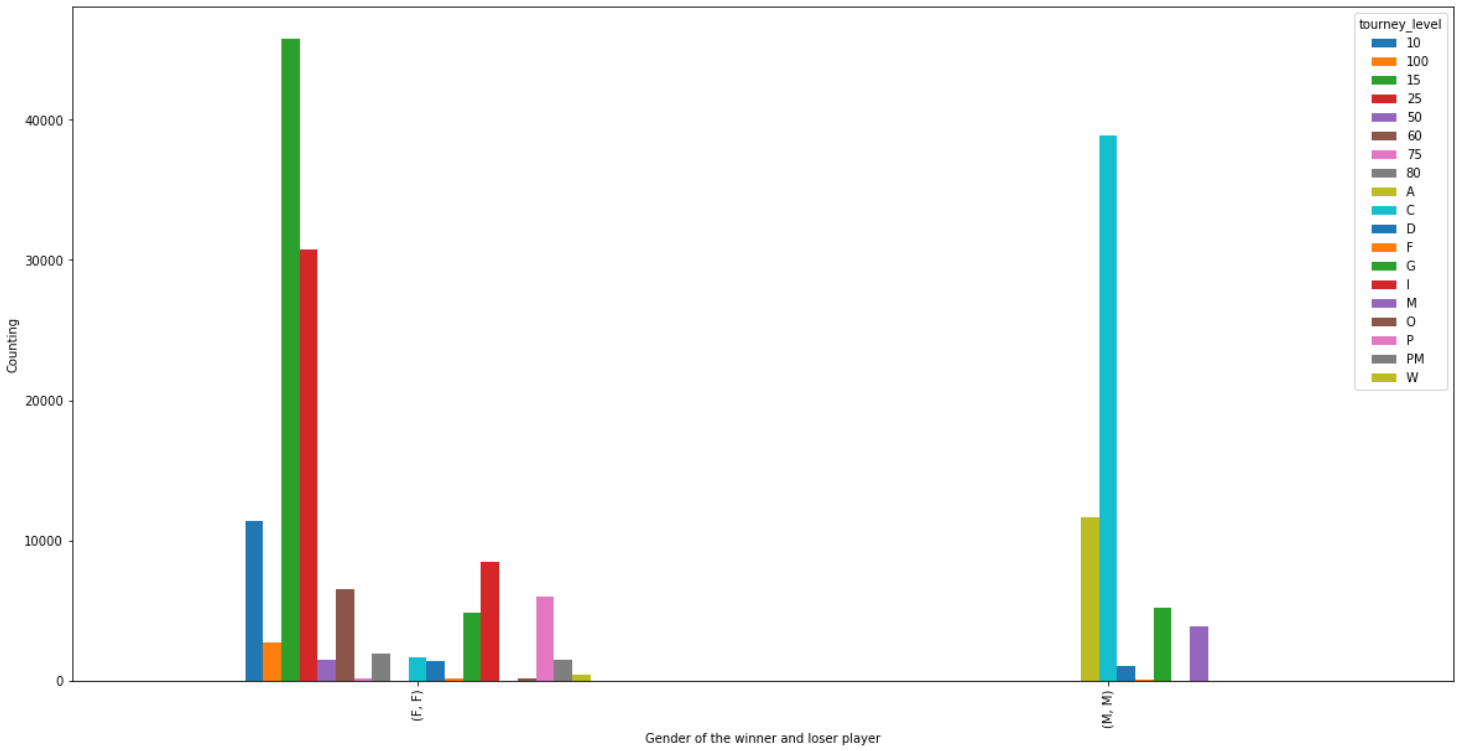

Fig.1.1.3.1: Distribution of turney_level values according to the gender of the player

For the ***height*** attribute there is a common consideration, firstly apply the mean according to the name of the player, but this replaces few of them so then compute the mean according to gender and nationality together. Later, there are yet some not resolved values, treat them with the mean of heights depending on only the gender. Since there could be the problem of adding an average value that is higher than a non-NaN value of an earlier year for the same player, this is not a wrong fact because in the following *preparation phase* it will be kept the max one.

To determine the missing ***age*** values of the players, it is needed to consider all the tournament years in which one could find a known age value of the same player. To achieve this aim, a new helpful attribute is introduced named ***birth***. It is obtained from the difference between the *tourney_date* with the *age* attribute of each player by saving only the corresponding year part, but since for a same year a player can have 2 assigned different ages if the two refer to one before and one after the date of birth, then take the smallest year's value. At the end there are 2 attributes denoting the same information, *age* and *birth,* which can be deduced one from each other and is redundant to keep them both, for this one of them has been dropped away, it has been opted for removing the *age* attribute and maintain the other one since is more convenient to have statical data. The *birth* attribute is an attribute that will never change while the other part is something dynamic that needs to be updated each year, and this dynamicity is useful to be saved only for attributes describing performance characteristics.

The last adopted consideration is that, if it is not possible to resolve an attribute by looking at the dataset but the only way is by using an external resource such as Google, then leave it as NaN. It is better to eventually resolve in this way for attributes related to the next data *preparation phase*.

Wrong Values

In this part, consider only the categorical values since they are checked according to their syntactic meaning. First of all there is a *pre-processing phase* in which it has adopted the strings' normalization attributes in which the special characters and punctuations are removed.

Regarding the ***tourney_name*** and their ***tourney_ids*** there are many (tournaments') names with the same id while at the same time there are many ids with different (tournament's) names too. By exploiting an association in one way, in which each ***tourney_id*** corresponds to a unique ***tourney_name*** the goal is to resolve syntattic errors in this last attribute. Initially extract the year from the ***tourney_id*** attribute and took only the part relating to the id (i.e.: 2021-xxxx becomes xxxx), from here analyze all the names by normalizing in case of double ones referring to the same one through the capitalization and transformed into the english correspondent place's name. In the end there are a total of 27 different **tourney_id** that refers to more than one **tourney_name**, which is a reduced set than the original one.

With ***player_name*** and their ***player_id***, there is the same problem (many names with many ids) and try the same considerations such as realizing the association of one name to a unique id and also the vice versa. There are no syntactic corrections to be applied on.

Regarding the ***nationalities****,* use a pycountry library in order to check if the code of each nation is written correctly, the analysis gives good results for all the attributes. Is noted that some players (3 to be precise) appear with multiple nationalities, following research with external sources it is checked which players have dual nationality and replace those which have the wrong nationalities with the correct ones.

For the ***hands*** of the players adopt the same procedure as in the previous phase, i.e. for each player take the type of hands with which it appears and then replace the replicated different values with the corresponding real ones.

The same for the ***gender*** of the player too, in order to have a clean and precise dataset, there is only one case and is resolved after looking on Google.

Outliers

Both the box-plot and scatter-plot graphs are used to visualize if there are some outliers for the numerical attributes, either individually or in a pairwise way to preserve some of their intrinsical relations. This makes it possible to decide differently for each type of situation. In some cases, doing the values' replacement has to be done in a careful manner since not all of them are outliers, they are real values even if they could appear not so common values.

In general, to approximate the correct outlier values, a function is implemented and it returns two values called: L and R. According to the distribution of the values, they are calculated in the following way:

- iqr = third_quartile - first_quartile
- L = first_quartile - 1.5*iqr
- R = third_quartile + 1.5*iqr

Notice that not all values out of the range formed by value L and R are considered outliers.

Players with an *outlier* ***height*** are only 3 which are solved using the official ATP website, to find the correct values to sign to them.

The ***_birth*** attribute has implied the reduction of a range in which are excluded those players older than 60 years of age, so are not admitted years of birth before 1960. This is justified by the fact that, as checked in the respective boxplot in Fig. 1.1.3.2, the points identifying the outlier values are those that go beyond this threshold. This is a case in which the points that go out the R and L values are not to be considered necessary outliers but could be just some not common values.

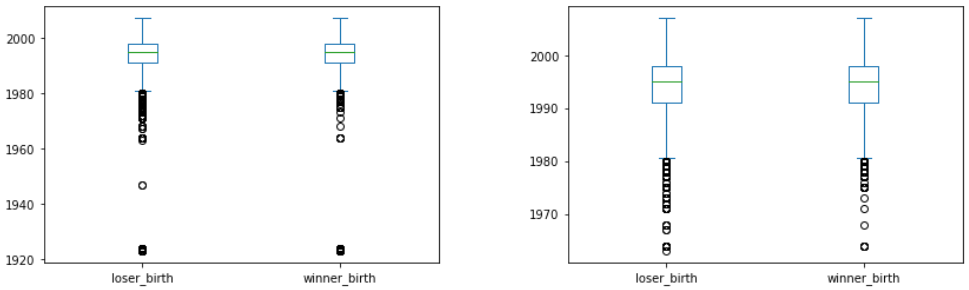

Fig.1.1.3.2 Boxplot of _birth before and after outliers' correction

Also the **_df** are resolved in the same way of **_birth** attributes.

With reference to the ***minutes*** attribute, consider its distribution graph Fig. 1.1.1.8, in which is highlighted a maximum admissible threshold of value 10 hours; there are a lot of outliers that are changed with the median.

Looking at Fig. 1.1.3.3 (left), the **ace** attribute has a coarse distribution that if considered by its own is not helpful to deduce some far away points from the others. Is needed to combine it with another attribute that can go in contrast for some of them even if it is not highly correlated with any of the other attributes. By a semantic meaning, it is considered the **svpt** attribute since a player cannot realize a distant number of aces than the number of svpt. According to Fig 1.1.3.3 (right) there is a slow increase of the number of aces with reference to those of the svpt. But some of them do not follow this behavior and so, for this reason are considered outlier values. They are cut off because they stay alone with reference to the others and are distant from the respective R border, so looking in the right part of the same figure, from the scatter plots are excluded these points.

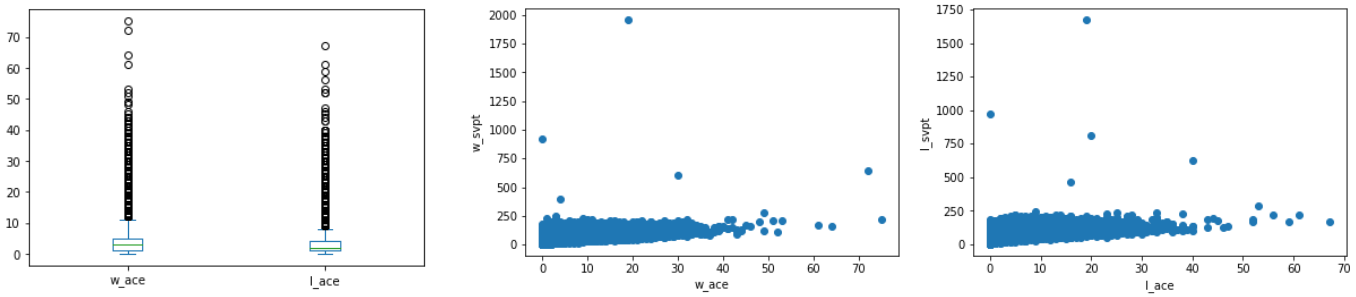

Fig 1.1.3.3: boxplot (left) and scatter plot (right) of _ace and _svpt

For the other attributes it has been considered their correlation meaning; regarding to **_1stIn** attributes, as already specified in the *correlation paragraph 1.1.2.*, it is strongly linked with **_svpt** attribute. For this reason a linear trend among them is attended and those points not following it are substituted with the corresponding R value.

The same considerations can be applied to the **_1stWon** with the **_1stIn** and **_2ndWon** with **1stIn,** with the aim of resolving the points that are far away from the general distribution.
Firstly, for both of them is considered the overall trend and cut those points that are far away from the others. In the second analysis is checked again the realized scatter plot graphs in Fig 1.1.3.4 and it is noted that something is not satisfied yet, as highlighted in red circles; in the first pair, it is semantically known that the values of **1_stWon** must be less than those of the corresponding **1_stIn** because it's impossible that the number of first serve points won is greater than first serve point made. For this reason, values of **1_stWon** that are in contrast with the other counterpart are changed according to a suited proportion. This last is realized by taking into account the total number of **1_stWon** with respect to the total number of **1_1stIn** and compute the amount of the particular value of **1_stWon** with the **1_1stIn**. The same results are applicable also to the second pair of attributes.

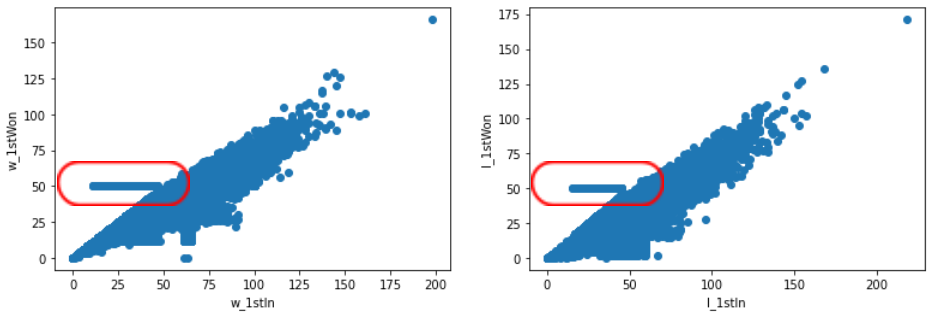

Fig 1.1.3.4: scatter plot for _1stIn and _1stWon, circled the _1stWon that are greater than the correspondent _1stIn

Since **rank** and **rank points** are very sparse because each player has a different assigned value then a box plot is not useful to resolve outliers, while it could be useful instead to analyze the trend of the hyperbola graph, referred before in Fig. 1.1.11 *in paragraph 1.1.* . It's possible to observe that no one point is out of this trend, so no one outlier is detected and resolved.

With reference to the remaining attributes of **_bpSaved** and **_bpFaced,** are considered firstly the boxplots and are replaced all the distant points with the right R and L value, then are checked also the scatter plots to verify the following of the linear trend since they have a high correlation too.

1.2 Description of the *Player* dataset

Starting from the observation of the matches, try to extract information about the characteristics and performances of each player, in order to create the player profile. The new dataset is composed by the following attributes:

- ***name***: name of the player
- ***hand***: hand of the player
- ***gender***: gender of the player
- ***ioc***: 3 character country code represented by the player
- ***birth***: year of birth of the player. It is computed by the difference between the tourney's year, in which the match has been played, and the age of the player. In the case there are 2 different ages we took the minimum since the other has been changed due to the day of the birth
- ***ht***: height of the player. It is the highest value
- ***minutes***: it is the mean of the played minutes
- **num_matches:** the total number of played matches, both either as a loser or a winner
- **num_matches_2016-2019** and **num_matches_2020-2021**: total number of played matches before and after the Covid disease. It is computed by summing the presence of a player among the matches among the 2 periods.
- **ratio**: it is the ratio between the won matches and the played ones.
- **ratio_2016-2019** and **ratio_2020-2021**: as above but splitting before and after the Covid disease
- ***perc_ace***: percentage of the total number of aces with respect to the total number of first serves made
- ***_1stwon_1stIn***: percentage of the total number of first serves won with respect to the total number of first serves made
- ***serv_won_tot_seve***: total number of won serves with respect to the the total number of services performed
- **bpFaced/*bpSaved***: total number of breakpoints faced/saved
- **perc_df:** percentage of double faults with respect to first serves made
- **perc_2ndwon:** percentage of second won with respect to second serves made
- **perc_v_ace**: percentage of aces suffered, calculated as the total number of opponents' aces with respect to the total number of opponents' first serves made against the player
- **perc_v_df**: percentage of double faults suffered, as above, calculated as the total number of opponents' double faults with respect to the total number of opponents' first serves made, against the player
- **perc_v_1stwon**: percentage of first serve return points won, calculated as the total number of opponents' first serves won with respect to the total number of opponents' first serves made against the player.
- **perc_df_2ndIn**: percentage of double faults suffered, calculated as the total number of opponents' double faults with respect to the total number of opponents' second serves made against the player. The amount of second services is calculated by the difference between **_svpt** and **_1_stIn.**

1.2.1 The Statistics, Distribution and Cleaning of the *Player* dataset

Firstly, the overall dataset has been reduced by dropping away players with less than 10 played matches since they are not useful for the wanted statistical purposes. In total the new table contains 3885 rows (one for each player) and 24 attributes.

Missing Values

The computed operations have obtained mathematical results transforming the null values into zeros. To better understand the distribution of the values, it has been consulted the graphical representation of all the numerical attributes (Fig. 1.2.1). It is noted that for most of them the value 0 is the predominant one and that

value represents both null and good values. Since it is not easy to distinguish the 2 situations it has been reasoned as follows: in the case in which a player has the majority of 0s among those attributes describing its skill performances (percentage of aces, double faults, won services and number of breakpoints*)*, then it has to be removed because it has not enough statistical information, otherwise it has to be kept. In the next Fig. 1.2.2, it is shown the distribution of attributes after these deletion, achieving an improved situation among the predominance on the 0s. This operation has not changed the overall distribution of the other values but just the predominant 0s, so this means that the predominant 0s were in the deleted rows only.
But not all the values have been influenced by the mathematical operations, so some null values remain and are those of the *ratio* and *birth* attributes. They have been treated as follows: all the null values of the *ratio* attribute have been changed into 0s, with what concerns the *birth* values it has been discovered only 2 and have been changed with the corresponding values through an ATP website. At the end the total number of players collected is 2120.

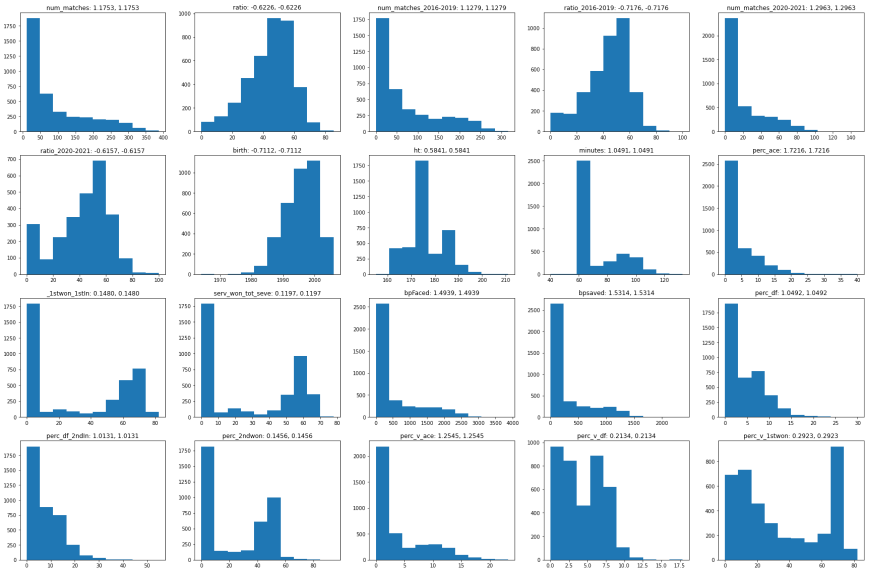

Fig.1.2.1 Before removing the players.

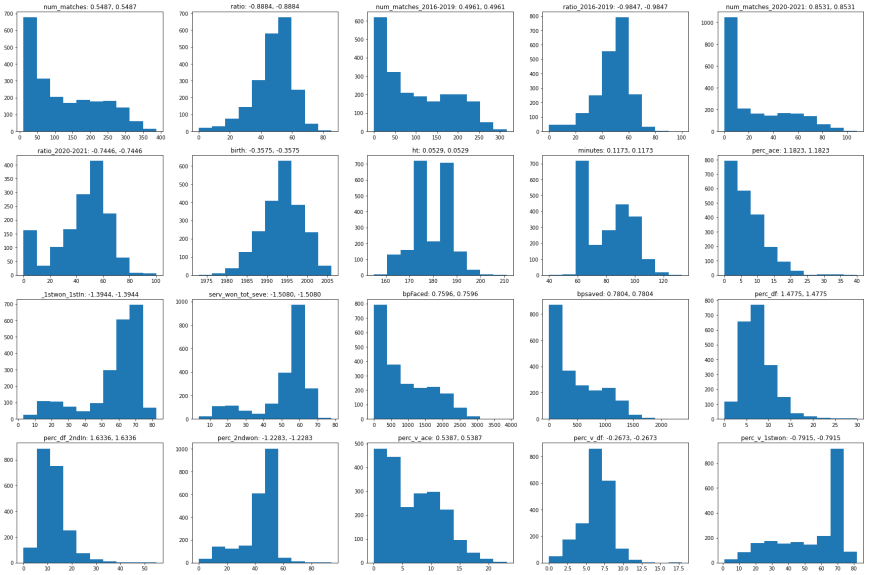

Fig 1.2.2 After have removed the players with the majority of values of 0

Outliers

This part is analysed by just viewing some boxplot graphs and noting that the more distant points are not errors but are useful values in determining the worst or better performances of the players, for this reason they are not treated. Moreover they are not too far away from the respective R and L values. Only the *ratio* attribute has been resolved, so in the case in which the general *ratio* is 0 then it has been removed, because either has not so many matches or is for sure an outlier.

Correlation

In Fig. 1.2.3 there is the correlation matrix of all the dataset, in particular there is a high correlation of 0.9 among the following attributes which grow proportionally with each other, for this they are redundant:

- save *_1stwon_1stIn and perc_2ndwon* and drop *serv_won_tot_serve*
- save *num_matches* and drop *num_matches_2016-2019, bpFaced*, *bpsaved*
- save *perc_df* and drop *perc_df_2ndIn*

The changes have achieved a total number of attributes from 24 to 19 elements.

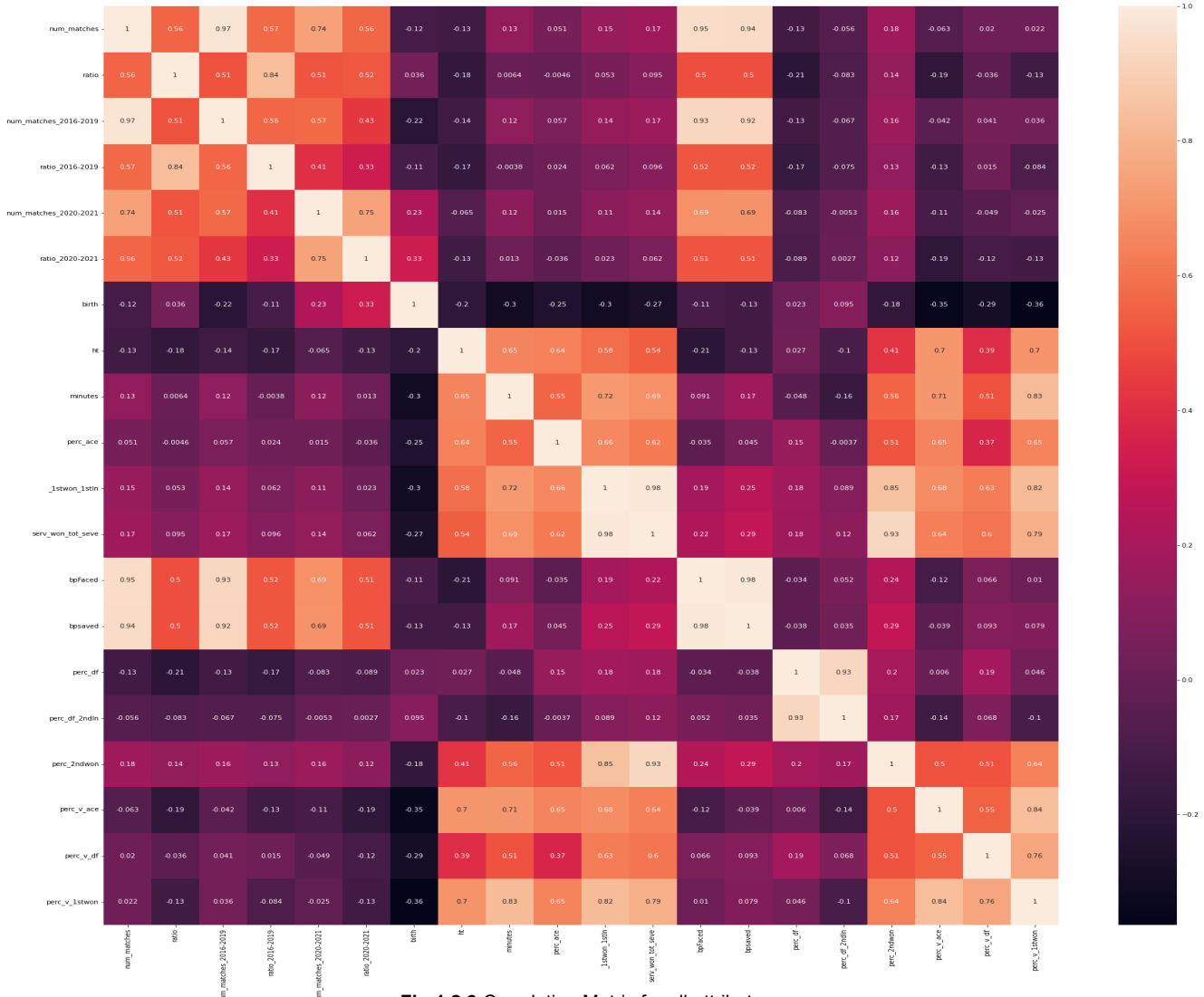

Fig 1.2.3 Correlation Matrix for all attributes

2. Data Clustering

2.1: Description of the *Tennis Player* dataset

The clustering analysis is performed on the *Tennis Player* dataset composed of 3906 rows (one for each player) and 19 attributes. They are used to analyse the behavior of each player over the years and to discover their performances.

2.2: Pre-processing

Before correctly applying the clustering algorithms, some pre-processing analysis is needed. This consists of 2 main activities:

- Remove the categorical data: they are not well suited data for this type of analysis, i.e *name*, *hand*, *gender*, *ioc* with remaining 15 numerical attributes. It is also an advantage of the analysis process since it reduces the dimensionality of the overall dataset.
- Normalization: the two metrics of MinMax and ZScore have been used obtaining similar results in both the cases. It has been opted for the MinMax approach.

2.3.1: K-Means

The goal is to partition and cluster the data into k groups through a center-based approach. The first decision is about a k and of each instance to the cluster according to the closest of a distance function. This process is repeated until a convergent situation.
Since the number of cluster decisions is not an easy task to be done manually then is used the **Elbow method** that is an heuristic in determining the optimal k. The interval of admissible k is between 2 (the minimum k cannot be either 0 or 1 otherwise could not cluster in a good manner) to 10. But not always this method is a good option since it does not work well if the data is not very clusterable, in this case there is a smooth curve not highlighting a knee in the graph. Then other scoring methods also can be used to exploit different metrics. In the following figures are shown the 3 metrics, denoting with a blue line the elbow function, a vertical dashed line the optimal k and a green line the time needed for each clustering of different k.

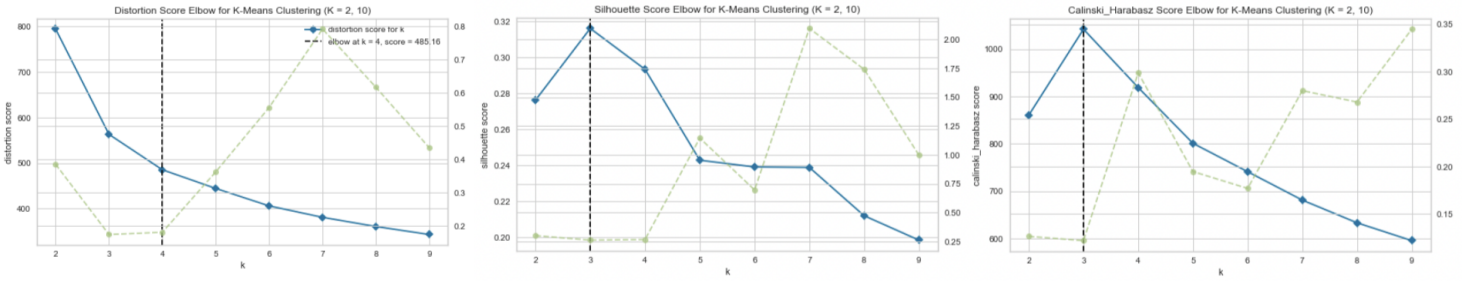

Fig. 2.3.1.1: from left to right the Elbow method with metrics: SSE, Silhouette, calinski score

The first is the *distortion* metric, based on the Sum of Squared Errors SSE with the best k=4, the second is the *average silhouette* computing the mean ratio of intra-cluster and nearest cluster distance with the best k=3, and finally the third is the *Calinski Harabasz* which computes the ratio of dispersion between and within clusters, again with the best k=3. Thanks to this graphical analysis the chosen k to be tested is 2, 3, 4 and also 8 since this is the standard initial value of KMeans. The best k selected is the one with the best Silhouette, that is 0.316, the same as the one shown with the Elbow function (Fig. 2.3.1.1 the middle one). It is known that the Silhouette values are in the interval [-1, 1] and the obtained one is closer to 0 than 1, a little more than half of the interval.
In the following section 2.3.4, will be shown the graphical comparison of the KMeans results with other clustering results.

2.3.2: DBScan

There are two parameters to be setted: *eps* and *number of neighbors*. To find the best combination of them, two vectors have been created, with the following values:

eps	0.005, 0.01, 0.05, 0.08, 0.1, 0.15, 0.20, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55
|neighbors|	3, 5, 6, 7, 8, 9, 10, 15, 16, 17, 18, 20

Fig 2.3.2.1 values of distinct combination

DBScan clusters have been implemented for all the possible combinations of these parameters, in total 168 choices. Then, it is created a pair consisting of each DBScan instance with its corresponding silhouette score, which is inserted into a list. To achieve the clustering validation step, this list is sorted in a decreasing way, according to the silhouette values. This is useful to analyze the best configurations, as shown in Tab. 2.1.

eps	|neighbors|	Silhouette	labels	|labels|
0.55	2	0.285	{-1, 0}	{ 2, 2118}
0.55	10	0.261	{-1, 0}	{ 3, 2117}
0.55	9	0.261	{-1, 0}	{ 3, 2117}
0.55	8	0.261	{-1, 0}	{ 3, 2117}
0.55	7	0.261	{-1, 0}	{ 3, 2117}
0.55	6	0.261	{-1, 0}	{ 3, 2117}
0.55	5	0.261	{-1, 0}	{3, 2117}
0.50	3	0.227	{-1, 0}	{7, 2113}
0.45	9	0.221	{-1, 0}	{ 30, 2090}
0.45	8	0.221	{-1, 0}	{30, 2090}

Fig 2.3.2.2 best results for each combination

Note that the best configurations have very similar parameters, in particular: the best *eps* is between 0.45 and 0.55 while |*neighbors*| is in a range of [2, 10], however the obtained *silhouette* is not so good because is more near to 0 respect to 1. With reference to the *silhouette* score, it is quite similar to that already obtained in the KMeans approach, and also later in the following Hierarchical method. Here with the DBScan manner, given the great amount of attempts and achieving always a low *silhouette*, then this could suggest that the data are maybe not so well clusterable. In any case, all configurations create only two clusters identified by two labels:

- label -1: represents the outliers and the DBScan method detects only few of them;
- label 0: conversely, represents the big cluster where each element is not marked as an outlier.

Since the silhouette is not so good, the points that are marked as outliers are not treated.

2.3.3: Hierarchical

The way in which the correct number of clusters is selected, it is made up of two approaches: *agglomerative* and *divisive*. The first one starts from many different points which gradually are joined together in a single agglomerative grouping, while the latter, conversely, starts from a single cluster common to all the points which is divided into smaller and smaller pieces. Between them it is adopted the agglomerative manner with 4 different types of linkage, which consider the similarity (or proximity) between any two clusters:

- *Single or Min*: that proximity is based on the two closest points of the different clusters
- *Complete*: that proximity is based on the two most distant points of the different clusters
- *Average*: that proximity is based on the average of pairwise (and not total to avoid large clusters) proximity between points in the two clusters
- *Ward*: that proximity is based on the increase in the Sum of Squared Error when two clusters are merged. Similarly to a group average.

In the following figures are shown the distributions of the agglomerative clusters analysis using dendrograms, in each of them is associated the respective default threshold which initially corresponds to the ⅘ of the size of each linkage matrix.

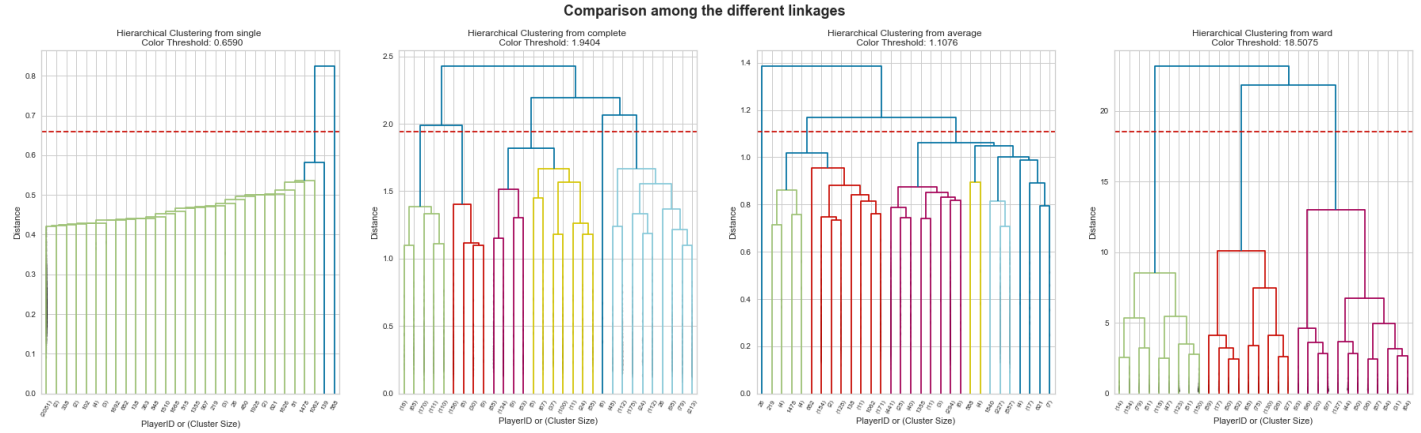

Fig 2.3.3.1 dendrograms for 4 type of linkages and threshold set to 4/5 of the size of each linkage matrix

The initially threshold of ⅘ is not a correct value for all the models; as a matter of fact the data are not well distributed with this cut since it does not maximize the overall distance of all the clusters i.e. see the distance between a cluster and another as in the first graph of Fig 2.3.3.1. For this reason in the following Fig 2.3.3.2. are shown the results of an algorithm that automatically chooses a threshold, for each of the metrics, by setting properly the right parameters. For instance the *single* metric has steps of 0.001 because it has a smaller distance than the others as the *ward* in which the step is 0.1 instead. In general, the algorithm starts from a low step and increases this value of the steps till it finds the maximum distance between groups of clusters, so before and after the unification, and finally at the end it returns the best threshold associated with the maximum distance found previously.
Are printed again the dendrograms with the best cut for each linkage.

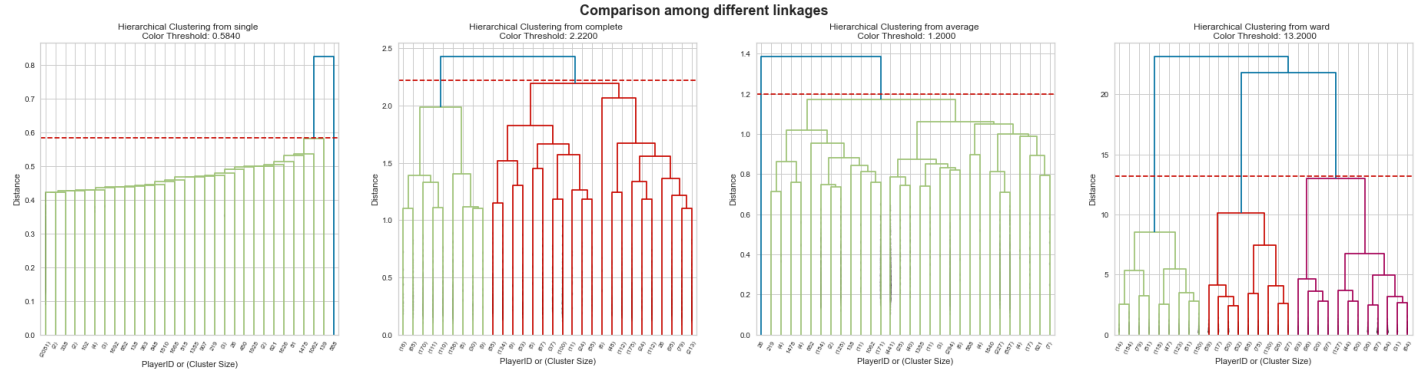

Fig 2.3.3.3 best dendrograms with correct threshold for 4 type of linkage

			Metrics		
Method	**Number of Clusters**	**Distance**	**Separation**	**Silhouette**	**Cophenetic Correlation**
single	2	0.58	0.499273	0.362651	0.301346
complete	2	2.20	1.661606	0.200896	0.587863
average	2	1.18	0.574748	0.265659	0.694167
ward	3	13.00	1.257283	0.285021	0.633060

Fig 2.3.3.2 Best clusters for the 4 type of linkages and each metric

In the table in Fig 2.3.3.2 are shown the automatic decided values, so for each method of linkage there is the best achieved number of clusters and the computed distance. Moreover, for the evaluation part there are the following types of measures:

- *Separation*: is the sum of the weights between nodes in the cluster and nodes outside the cluster;
- *Silhouette*: combines ideas of both cohesion and separation; the best result
- *Cophenetic correlation*: measures how faithfully a dendrogram preserves the pairwise distances between the original unmodeled data points.

According to the Cophenetic correlation and silhouette the best model is obtained with the ward method, while looking only at the silhouette the best model is obtained with the single method.

2.3.5: Optional clustering algorithm: XMeans

To implement this clustering method, the pyclustering library is used. The chosen parameters are those for the interval of admissible values that are from 2 till 10, and the tolerance level which allows to decide how much this splitting can be admissible or not. The histogram in Fig 2.3.5.1 shows the distribution of players inside each cluster, most of them are equally distributed apart for only one that has the most points inside it. The overall graphical clustering results are discussed in the next paragraph.
The silhouette obtained is similar to previous clustering methods: 0.27, this suggests that also in this case the data are not well clustered.
.

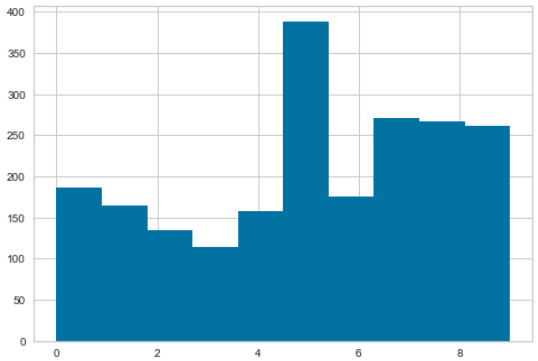

Fig 2.3.5.1: Number of players for each cluster

2.3.5: Clustering comparisons

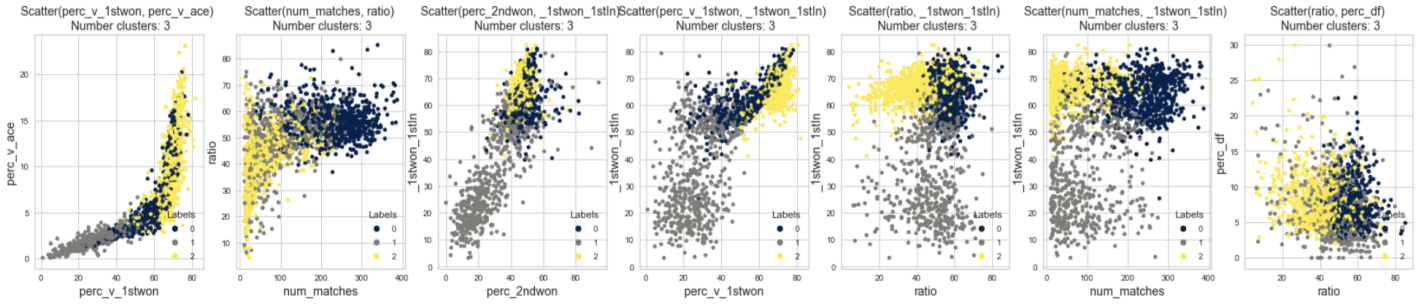

Fig 2.3.5.1 KMeans

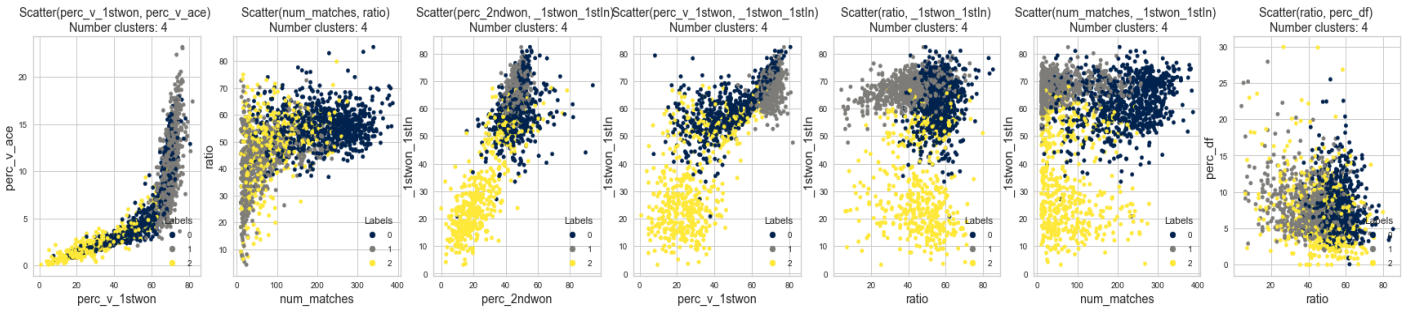

Fig 2.3.5.2 Hierarchical with ward method

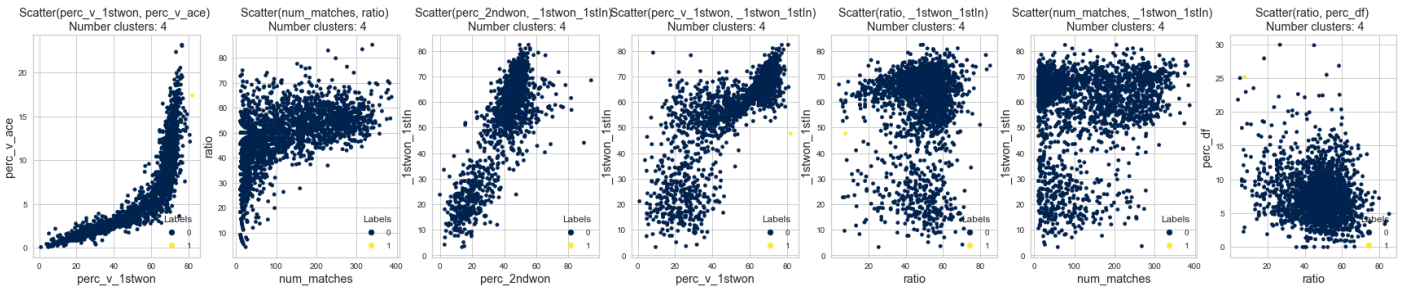

Fig 2.3.5.3 Hierarchical with single method

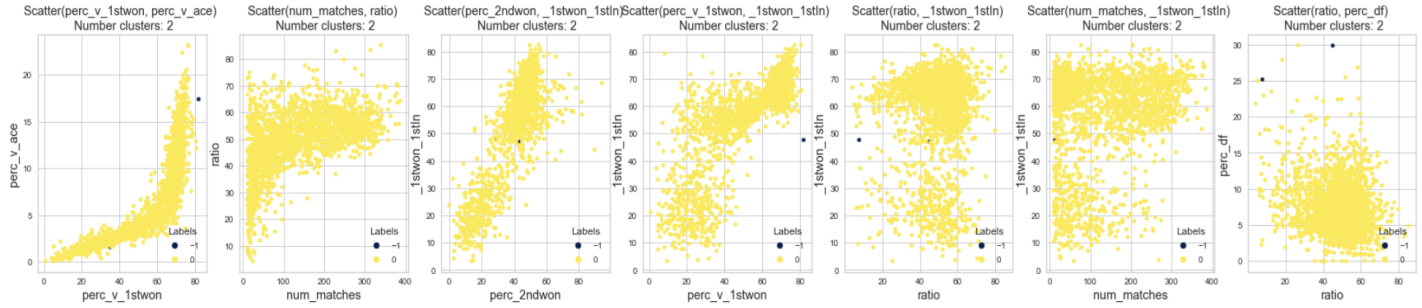

Fig 2.3.5.4 DBScan

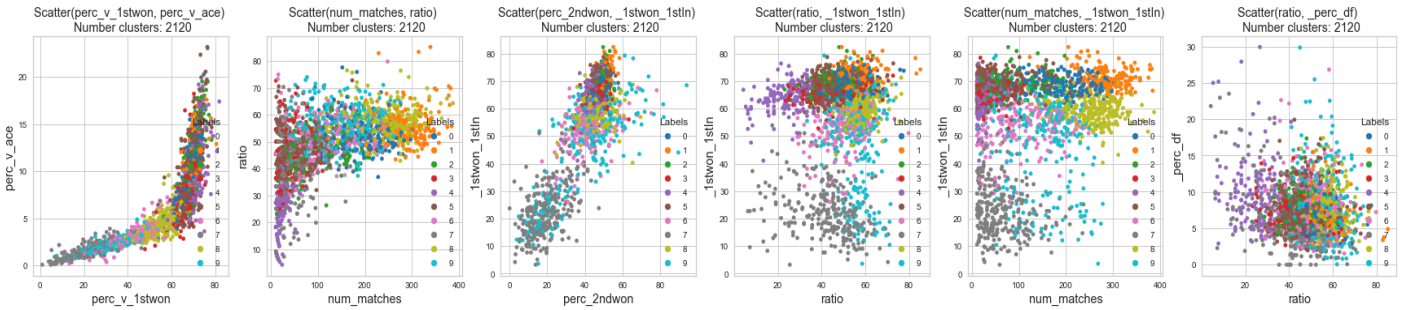

Fig 2.3.5.5 XMeans

Comparing Fig 2.3.4.1 of KMeans and Fig 2.3.5.2 of Hierarchical with the ward method, it is concluded that they have the same behavior; they both divide the data among 3 distinct clusters. This was something already highlighted previously in the respective description part but here it is highlighted also graphically. Moreover, the distribution on the 3 discovered clusters are of the same manner.

In reference to DBscan in Fig 2.3.5.4 and Hierarchical Fig 2.3.5.3 with the single method, there is a common behavior. Also in this case the two approaches give very similar results with only one big cluster and some few points sparse in the graph. They are good values since they highlight small outlier values.

The last figure 2.3.5.5 represents the clusters obtained with X-means, but it's not possible to distinguish graphically all clusters created.

3. Classification

In this task, the aim is to perform a predictive analysis in order to classify the players in two categories on the basis of their rank attributes. Since in the players' table, built above from the preparation part, there is no one attribute correlated to the rank, then all the ranks of each player have been extracted from the initial matches' table. Then the mean of the ranks has been calculated and finally it has been associated with the corresponding player in the players' table. After this operation only 40 players of 2120 remain without a rank value, so these elements are removed from the players' table in order to perform the classification task only to players that have a rank assigned to them. Players with no rank are not removed at all but kept separately in another table, and after performing the classification on the previous ones, the model obtained is used to predict labels of players with no rank too.
To assign a label, players are sorted by ranks and divided into two categories: high-level players and low-level players. The first 25% represents the high-level players and are identified by label 0, because it has been taken the mean of the players' ranks, they are the players that maintain the best results over time, vice versa, the other 75% represents the low-level players and are identified by label 1. Note that this splitting has provided an unbalanced distribution of labels, so in some classifiers a *class weights* parameter has been used.
Since not all the classifiers can predict in a difficult manner if the data are not well presented, for the majority of them is necessary the discretization step, by transforming the categorical attributes into the corresponding numerical ones.
The dataset is divided in development (75%) and test set (25%) maintaining the percentage of labels in both the two sets. For each classifier, a grid search is implemented and the best model is selected according to *f1_score* metric. However other evaluation methods are used to have a better view of results, in particular are reported:

- Classification report through *accuracy*, *precision*, *recall* and *f1_score* metrics
- Roc-Curve, not possible to be computed in KNN and Radius-Neighbors
- Confusion matrix

Table 3.1 describes all classification methods used and results of *f1_score*, for both high-level and low level players, using the best model returned by grid search with **8-fold cross validation**.

	Training Set f1_score		Test Set f1_score	
Classifier	**high level**	**low level**	**high level**	**low level**
Decision Tree	0.87	0.96	0.86	0.95
Bayesian classifier	0.74	0.89	0.72	0.87
Neural Network	0.84	0.95	0.82	0.93
K-Near Neighbors	1.00	1.00	0.85	0.95
Radius-Neighbors	1.00	1.00	0.23	0.87
SVM	0.88	0.96	0.89	0.96
Rule based	0.80	0.92	0.79	0.91
Ada Boosting	0.93	0.97	0.85	0.95
Random Forest	0.91	0.97	0.85	0.95

Tab 3.1 Global models evaluation

Nine different methods of classification are implemented using several libraries, in particular sklearn for most of them except the Neural Network realized with tensorflow and the Rule based with wittgenstein.

The first thing to pay attention to is that the obtained *f1_score* of high-level players is always lower than that of the low-level counterpart. This is due to two facts: labels are unbalanced since the players with high level labels are only ¼ of the total data, and mainly because the high level labels are very few elements, so the classifier has not so many samples to fit these labels well. Where possible, to overcome the first issue, as in the **Decision Tree classifier** and **Random Forest**, it has been opted to add some weights inside the grid search till reaching a well performance combination.

The worst models obtained are **K-Near Neighbors** and **Radius-Neighbors.** To find the best K for KNN, an interval between [$\sqrt{n}$-10, $\sqrt{n}$+10] is chosen where n is the train set size. The grid search for Radius-Neighbors is realized through several combinations of the parameters, in particular: *the radius, the metric, the algorithm types and the weights*. The reason why they are not behaving well is, even if the *f1_score* on the training set is the highest conversely in the test set is the worst, so it is an overfitting scenario.

With reference to the, **Neural Network** and the **Rule Based** are considered behaving quite well.
The first classifier is a net composed of only 1 layer more than the output layer, with an uniform kernel initializer for both, the *linear* activation function for the output layer and the *relu* activation function for the other layer. It has a grid search composed of: *n_neurons1, activation1, optimizer, activation_out, loss* parameters, each of them with only one possible choice, the one that, after several attempts, has reached the best performance.
The second classifier has a grid search setting the parameters of *prune_size* and the *k*.
They all have a high and similar *f1_score* both in the training and test, greater than 0.90 in the low-level part and greater than quite 0.80 for the high-level part. So it does not go into an overfitting scenario.

The best models are **Decision Tree**, **Random Forest, Ada Boosting** and **SVM**.
The grid search for the first classifier combines the following parameters: *max_depth, min_samples_split, min_samples_leaf, criterion, splitter, max_features, class_weight.* It has been performed on an overall 4000 possible combination of models.
The second has as a randomized and not a simple grid search, with the same type of parameters of the Decision Tree classifier and another that is the *bootstrap*.
The third one uses as a base estimator the Decision Tree and the other needed are the same used normally by default.

The fourth classifier, since it has a high complexity, is used with a small number of several combinations of parameters.
They all don't go into overfitting, and have the highest *f1_score* in both the training and test set, both in high-level and low-level parts.
For them is shown in the following graphs of Fig 3.2 their ROC curve. The curves are very similar between them; these metrics confirm the goodness of the four classifiers because the elbow is near to 1.0 of y-axis.

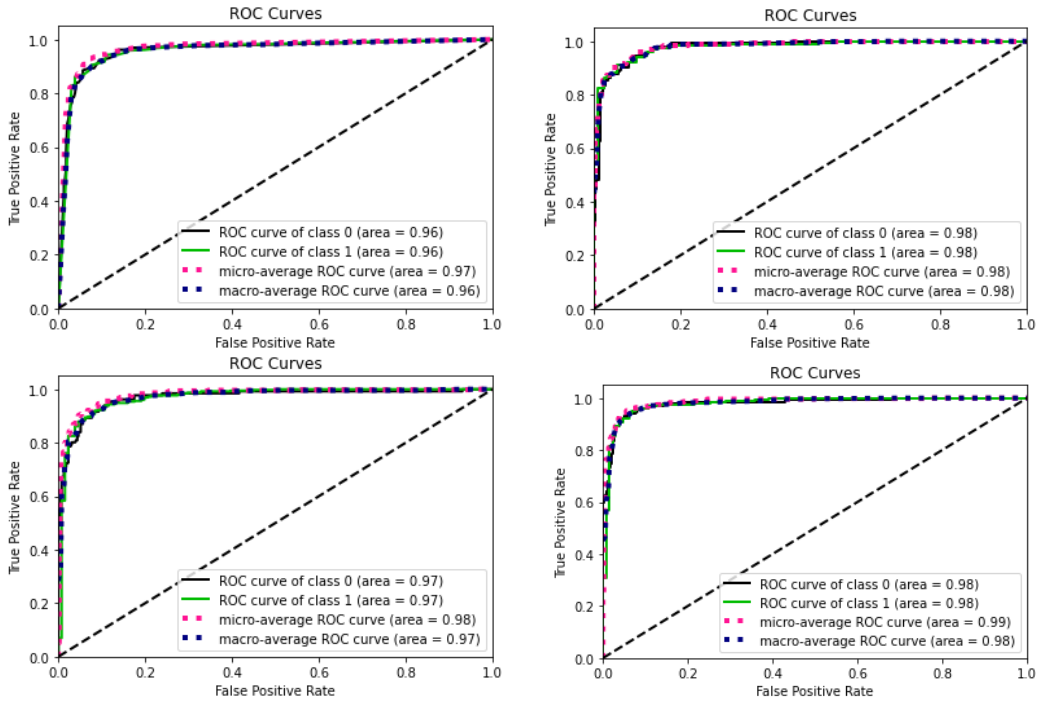

Fig 3.2 Decision Tree, RandomForest, Adaboosting, SVM

By comparing the ROC curve of all the applied classifiers can be deduced as follows. The ones with the best performance are those closer to the y-axis which are the same that confirm the previous observed conclusions. Instead the worst one is the Radius-neighbors. Also the remaining others agree with what has already been explained. For what concerns the players that were firstly excluded with the aim of achieving the classification tasks, the models are applied to them. All classifiers assign the label 1 to these players that are without a rank value, except for KNN and RNN, this could confirm again the fact that the models obtained with these classification methods are not so good.

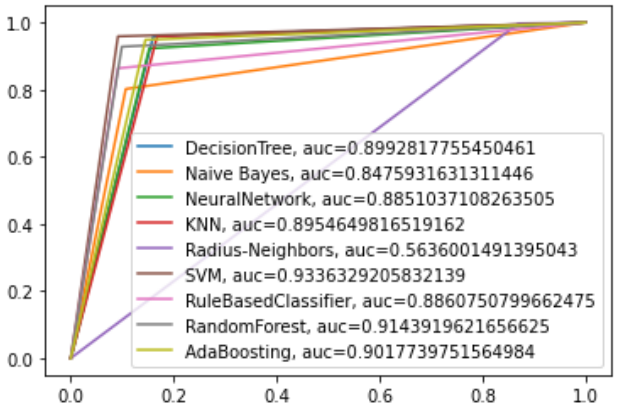

Fig 3.3 ROC curve for all classifiers