PREDICTIVE METHODS OF DATA MINING COURSE'S PROJECT

REPORT

Professors
Carina Albuquerque
Joana Rafael
Lara Oliveira
Ricardo Santos

Group 17
Alexandre Dionísio Esteves M20221293
Raimundo Mujica Costa M20221342
Simão Pedro Acciaioli Gouveia Dos Santos M20220561
Tomás Caldeira Cardoso Soares Esteves M20220377
Turay Filipe de Melo M20220567

Abstract

This report covers the process that was carried to achieve a classification model for *WCA* athletes' sports competitions' participations. The processes described are the process of accessing on the quality of the datasets and their condition, the dataframes pre-processing, and the model training and assembly, based on a pre-selected set of variables.

In the end, the evaluation measures for the model prototype are displayed to serve as a basis of discussion.

Table of Contents

Table of Illustrations

The Company

Introduction to We Are the Champions (WAC) Company

WCA (We Are the Champions) is a sports technology company that uses machine learning to supply accurate athlete performance predictions. They analyse diverse data sources, including sports organizations and performance statistics, to uncover patterns and refine prediction algorithms.

WCA serves various customer segments, such as professional sports organizations, sports enthusiasts, media, and the betting industry, offering forecasting and data analytics for a competitive advantage. Their revenue streams come from subscription services, data licensing, and partnerships with sports betting platforms.

WCA focuses on customer relationships and continuous innovation to maintain the industry leadership.

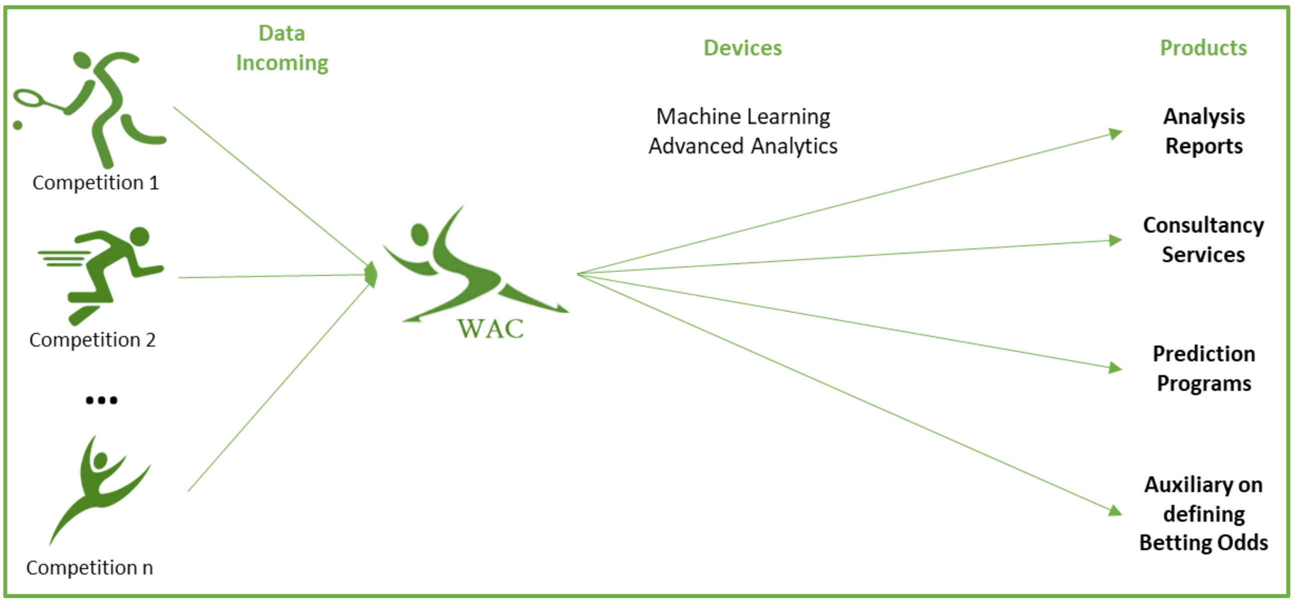


FIGURE 1 WCA BUSINESS MODEL

Company's Problem Scenario

The objective of the task which was handled to our group was to leverage a set of algorithms which study various factors and indicators of the athletes' performances in such a way that, in the end, we would be able to achieve a "trustful forecast" on an athlete's emergence as the winner or the loser in each competition.

We aim to help the way predictions are made within the sports industry, contributing to *WCA*'s more accurate vision that may help the company on assembling their products design.

Methodology Introduction

On assembling a predictive model, our team had to achieve a general mindset on the concept of the model's conception. Only after that we would be able to develop our prototype.

We briefly explain each one of the steps we will cover during this report.

1. **Package Installing and Defining Auxiliary Functions**

We present the process of installing the necessary packages and libraries required for our data analysis tasks. We also and implement auxiliary functions that assist in data processing, visualization, and analysis, enhancing the efficiency of our workflow.

2. **Data Integrity and Requisites Confirmation**

We performed data validation and verification processes to confirm that the data meets the required standards and requirements. This step helps us identify any missing or inconsistent data that may affect the accuracy of our analysis. In this step we will be also defining two different partitions of *WCA*'s train dataset.

3. **Data's First Statistics and Data Visualization**

We made an initial exploration of the dataset by calculating basic statistics such as the mean, the median, and the standard deviation. Additionally, supported by some visualization objects, we try to think on the logical possible relations among the variables.

4. **Data Pre-Processing**

We removed the outliers, and we normalized the dataframes. By pre-processing the data, we ensured that the data was in a suitable format for modelling.

5. **Second Data Understanding and Analysis**

We comment on the phase where we employed algorithms that result in a dimensionality reduction. This enables us to extract essential features and improve the efficiency of our models.

6. **Predicting the Outcome of an Athlete (Model Assembly and Evaluation)**

We run a series of pre-configured algorithms to achieve a final model prototype.

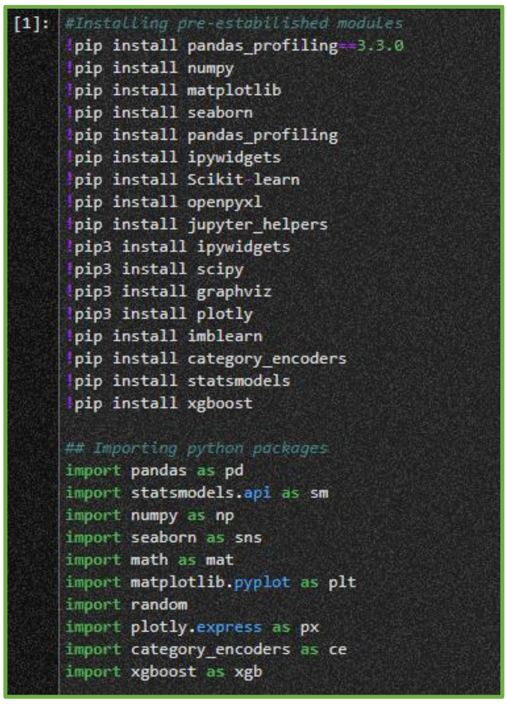
7. **Prediction on test dataset**

We apply the same pre-processing early used on *WCA*'s *train* dataset on the *test* dataset and run the model prototype on it.

Package Installing and Defining Auxiliary Functions

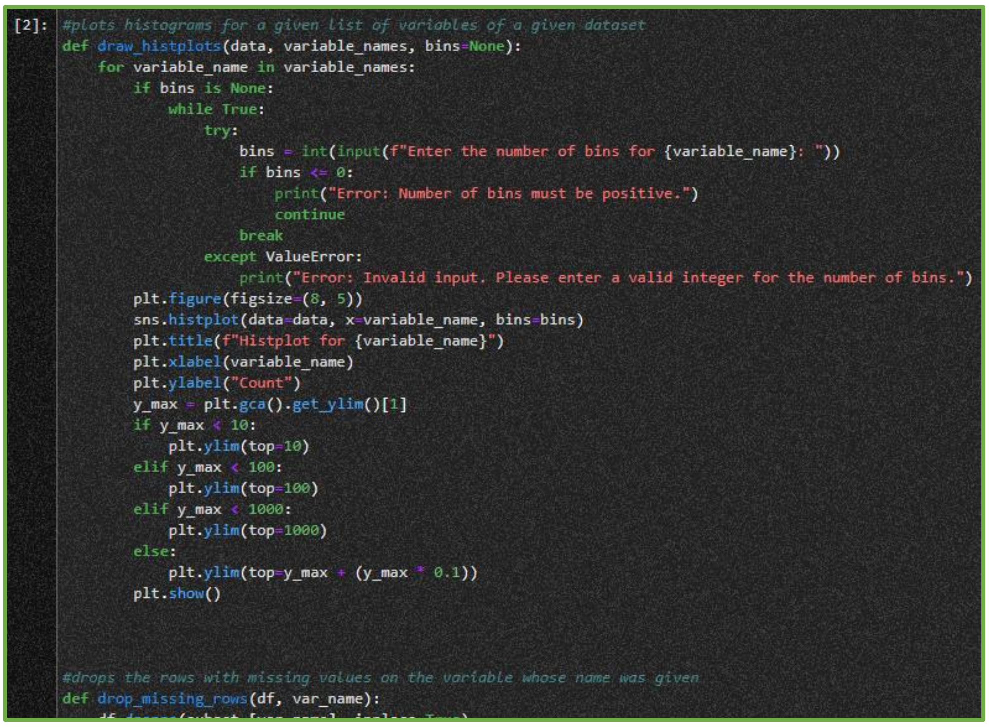
Before we start, we inquired our task about the resources it required for a proper and complete function. For that so, we had to go through two diverse sources: modules and auxiliary functions.

With the proper use of the right modules, we take advantage of using already defined and tested functionalities which are the core of the algorithm designing process. The second allows us to automatize a set of extra tasks that will help us throughout the project.



```
[1]: #Installing pre-estabilished modules
     !pip install pandas_profiling==3.3.0
     !pip install numpy
     !pip install matplotlib
     !pip install seaborn
     !pip install pandas_profiling
     !pip install ipywidgets
     !pip install Scikit-learn
     !pip install openpyxl
     !pip install jupyter_helpers
     !pip3 install ipywidgets
     !pip3 install scipy
     !pip3 install graphviz
     !pip3 install plotly
     !pip install imblearn
     !pip install category_encoders
     !pip install statsmodels
     !pip install xgboost

     ## Importing python packages
     import pandas as pd
     import statsmodels.api as sm
     import numpy as np
     import seaborn as sns
     import math as mat
     import matplotlib.pyplot as plt
     import random
     import plotly.express as px
     import category_encoders as ce
     import xgboost as xgb
```

FIGURE 2 INSTALLING AND IMPORTING MODULES

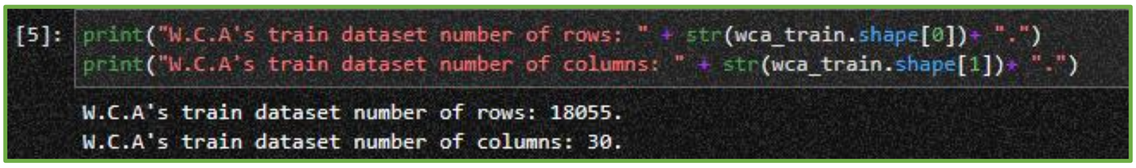


```
[2]: #plots histograms for a given list of variables of a given dataset
     def draw_histplots(data, variable_names, bins=None):
         for variable_name in variable_names:
             if bins is None:
                 while True:
                     try:
                         bins = int(input(f"Enter the number of bins for {variable_name}: "))
                         if bins <= 0:
                             print("Error: Number of bins must be positive.")
                             continue
                         break
                     except ValueError:
                         print("Error: Invalid input. Please enter a valid integer for the number of bins.")
             plt.figure(figsize=(8, 5))
             sns.histplot(data=data, x=variable_name, bins=bins)
             plt.title(f"Histplot for {variable_name}")
             plt.xlabel(variable_name)
             plt.ylabel("Count")
             y_max = plt.gca().get_ylim()[1]
             if y_max < 10:
                 plt.ylim(top=10)
             elif y_max < 100:
                 plt.ylim(top=100)
             elif y_max < 1000:
                 plt.ylim(top=1000)
             else:
                 plt.ylim(top=y_max + (y_max * 0.1))
             plt.show()


     #drops the rows with missing values on the variable whose name was given
     def drop_missing_rows(df, var_name):
```

FIGURE 2 ONE OF THE DEFINED FUNCTIONS, DRAW_HISTPLOTS

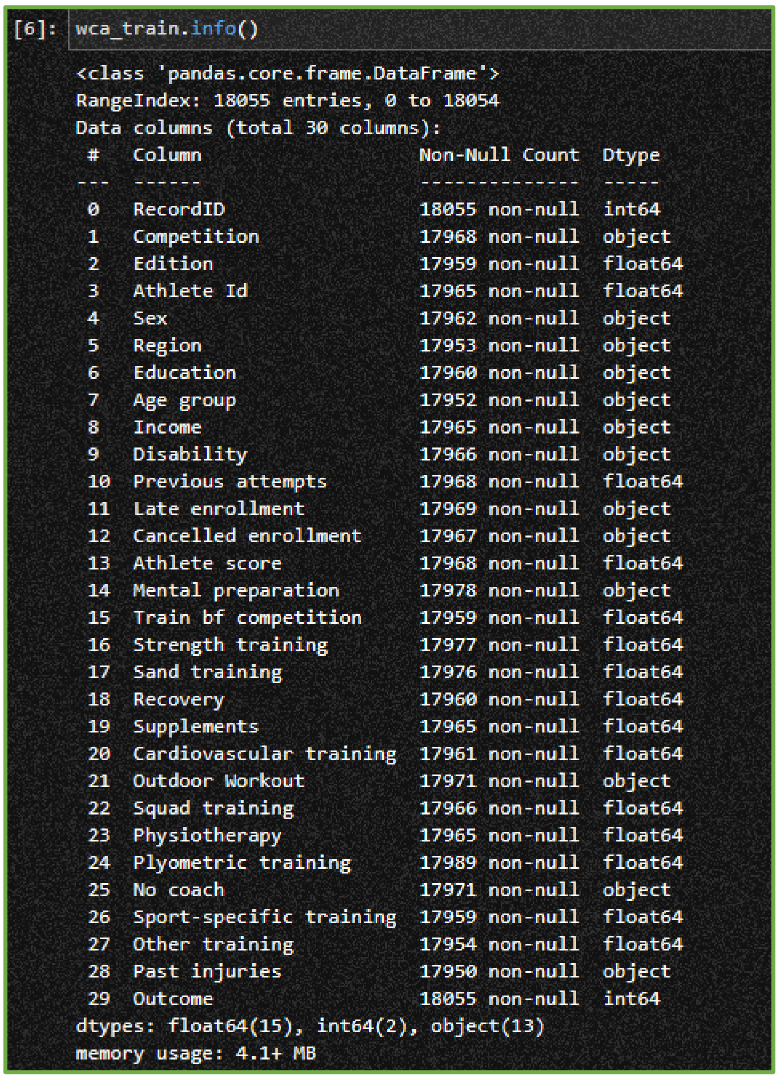
Loading and Checking on the condition of WCA's dataset

After loading our *WCA* **train** dataset we check on its initial conditions, namely size and variables.



```
[5]: print("W.C.A's train dataset number of rows: " + str(wca_train.shape[0])+ ".")
     print("W.C.A's train dataset number of columns: " + str(wca_train.shape[1])+ ".")

     W.C.A's train dataset number of rows: 18055.
     W.C.A's train dataset number of columns: 30.
```

FIGURE 3 INITIAL DATASET SIZE



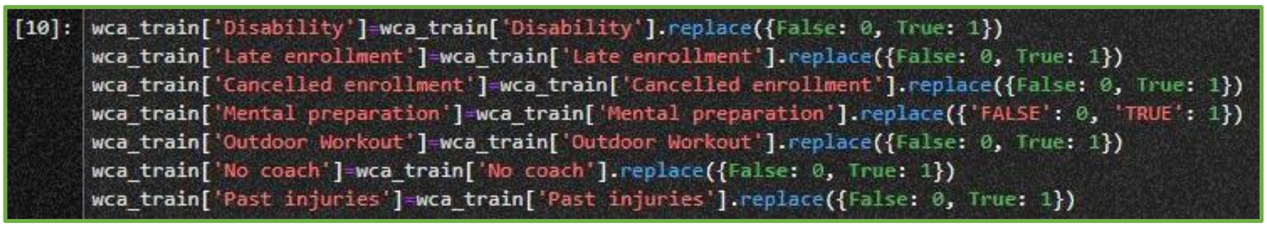
```
[6]: wca_train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 18055 entries, 0 to 18054
     Data columns (total 30 columns):
      #   Column                  Non-Null Count  Dtype
     ---  ------                  --------------  -----
      0   RecordID                18055 non-null  int64
      1   Competition             17968 non-null  object
      2   Edition                 17959 non-null  float64
      3   Athlete Id              17965 non-null  float64
      4   Sex                     17962 non-null  object
      5   Region                  17953 non-null  object
      6   Education               17960 non-null  object
      7   Age group               17952 non-null  object
      8   Income                  17965 non-null  object
      9   Disability              17966 non-null  object
      10  Previous attempts       17968 non-null  float64
      11  Late enrollment         17969 non-null  object
      12  Cancelled enrollment    17967 non-null  object
      13  Athlete score           17968 non-null  float64
      14  Mental preparation      17978 non-null  object
      15  Train bf competition    17959 non-null  float64
      16  Strength training       17977 non-null  float64
      17  Sand training           17976 non-null  float64
      18  Recovery                17960 non-null  float64
      19  Supplements             17965 non-null  float64
      20  Cardiovascular training 17961 non-null  float64
      21  Outdoor Workout         17971 non-null  object
      22  Squad training          17966 non-null  float64
      23  Physiotherapy           17965 non-null  float64
      24  Plyometric training     17989 non-null  float64
      25  No coach                17971 non-null  object
      26  Sport-specific training 17959 non-null  float64
      27  Other training          17954 non-null  float64
      28  Past injuries           17950 non-null  object
      29  Outcome                 18055 non-null  int64
     dtypes: float64(15), int64(2), object(13)
     memory usage: 4.1+ MB
```

FIGURE 4 INITIAL DATASET VARIABLES

Data Integrity and Requisites Confirmation

Fixing Variables' Data Types

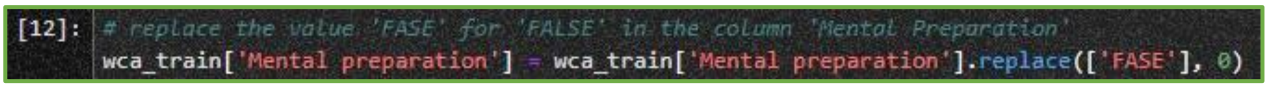
We immediately addressed the replacement of Boolean variables vales presented as "False / True" and "FALSE / TRUE" to zero or one (0 / 1). We also change variable *'Sex'* values "F" and "M" to this scale.



```
[10]: wca_train['Disability']=wca_train['Disability'].replace({False: 0, True: 1})
      wca_train['Late enrollment']=wca_train['Late enrollment'].replace({False: 0, True: 1})
      wca_train['Cancelled enrollment']=wca_train['Cancelled enrollment'].replace({False: 0, True: 1})
      wca_train['Mental preparation']=wca_train['Mental preparation'].replace({'FALSE': 0, 'TRUE': 1})
      wca_train['Outdoor Workout']=wca_train['Outdoor Workout'].replace({False: 0, True: 1})
      wca_train['No coach']=wca_train['No coach'].replace({False: 0, True: 1})
      wca_train['Past injuries']=wca_train['Past injuries'].replace({False: 0, True: 1})
```

FIGURE 5 REPLACING TRUE / FALSEWITH O AND 1 VALUES

Within this process, we found that variable *'Mental preparation'* displayed a value "FASE". As this was a single entree, we assumed it was a typo and should be "FALSE".



```
[12]: # replace the value 'FASE' for 'FALSE' in the column 'Mental Preparation'
      wca_train['Mental preparation'] = wca_train['Mental preparation'].replace(['FASE'], 0)
```

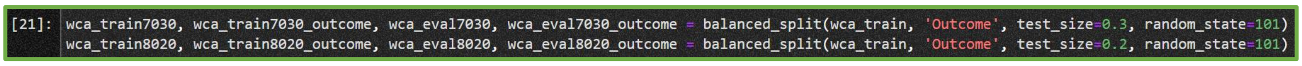
FIGURE 6 FIXING VARIABLE 'MENTAL PREPARATION' TYPO

We changed the type of the integer variables to float, to try to avoid future problems in the modelling section.

Defining the Sub datasets

We split our initial train dataframe based on two different balanced *'Outcome'* partitions for the percentage of train / evaluation sub datasets: 70/30 and 80/20. Additionally, we set variable *'RecordID'* as the index of the dataframes.

We end up with 4 new series objects.



```
[21]: wca_train7030, wca_train7030_outcome, wca_eval7030, wca_eval7030_outcome = balanced_split(wca_train, 'Outcome', test_size=0.3, random_state=101)
      wca_train8020, wca_train8020_outcome, wca_eval8020, wca_eval8020_outcome = balanced_split(wca_train, 'Outcome', test_size=0.2, random_state=101)
```

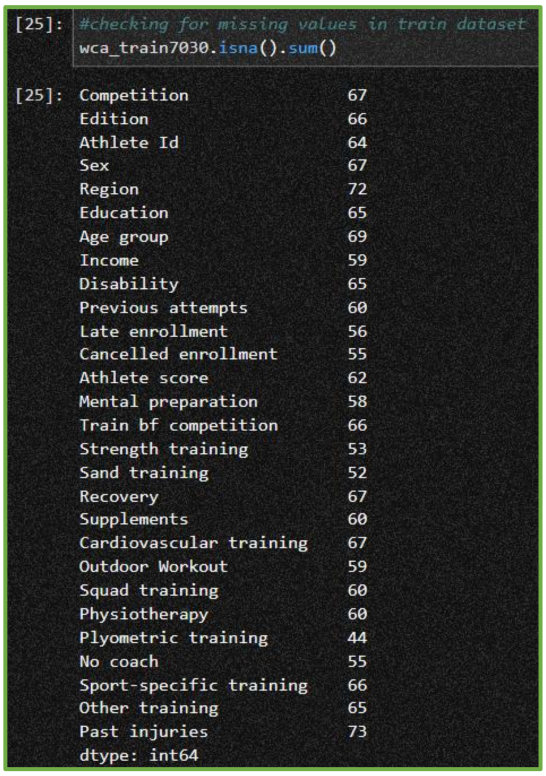
FIGURE 7 NEWLY DEFINED SUB DADATASETS

Dealing with blank/ missing values

Although early, the proper handle of missing values in a dataset is a crucial working stage: missing data can lead to biased or inaccurate results, affect the integrity of statistical analyses, and hinder the performance of machine learning models.

We analysed the missing values per row and found that the highest number of missing values within a record was three. Although we were unsure if all twenty-nine variables would be used, we ultimately concluded that filling three missing values within a record would not significantly impact the data's integrity or accuracy.

We noticed there were missing values within every variable of the datasets.



```
[25]: #checking for missing values in train dataset
      wca_train7030.isna().sum()

[25]: Competition                 67
      Edition                     66
      Athlete Id                  64
      Sex                         67
      Region                      72
      Education                   65
      Age group                   69
      Income                      59
      Disability                  65
      Previous attempts           60
      Late enrollment             56
      Cancelled enrollment        55
      Athlete score               62
      Mental preparation          58
      Train bf competition        66
      Strength training           53
      Sand training               52
      Recovery                    67
      Supplements                 60
      Cardiovascular training     67
      Outdoor Workout             59
      Squad training              60
      Physiotherapy               60
      Plyometric training         44
      No coach                    55
      Sport-specific training     66
      Other training              65
      Past injuries               73
      dtype: int64
```

FIGURE 3 MISSING VALUES PER VARIABLE

Depending on the case, we immediately began to think on the right way to address these issues, as we may see in below. We were supported by histogram plots and variables' statistics.

Variables	Strategy Employed	Reason
'Athlete ID'	We excluded records with missing values in the 'Athlete ID' variable.	Ensure general independence between athletes' attempts and avoid incorrect assumptions about sports performance, such as assuming no room for improvement or attributing poor performances solely to the athlete's abilities. **(*)**
'Previous attempts', *'Sand training'*, *'Team training'*, *'Plyometric training'*, *'Sport specific training'*, *'Other training'*	Fill in the missing values of these variables with their median values.	"Well behaviour" of the variables (low difference between mean and median; low standard deviation)
'No coach'	Variable was eliminated	Precarious number of athletes with "No Coach".
remaining variables	Filled in using KNN Imputer	Variables diversity of values but low consistency

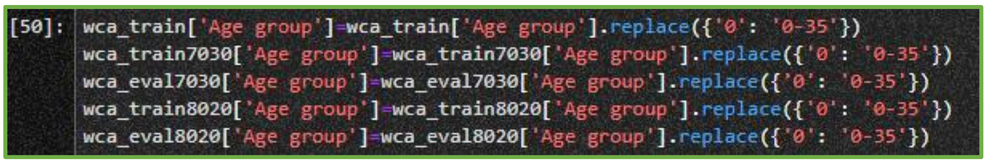
FIGURE 4 STRATEGIES FOR FILLING IN MISSING VALUES PER VARIABLE

(*) After this records droppage we re-checked the datasets' size.

Checking on the Values Coherence

We prioritized addressing the variables' values imprecisions.

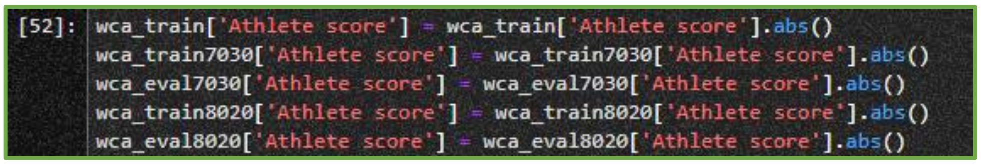
We noticed odd entries within the training dataset on variable 'Age group' with value 0, so we replaced the zeros for value '0-35'.



```
[50]: wca_train['Age group']=wca_train['Age group'].replace({'0': '0-35'})
      wca_train7030['Age group']=wca_train7030['Age group'].replace({'0': '0-35'})
      wca_eval7030['Age group']=wca_eval7030['Age group'].replace({'0': '0-35'})
      wca_train8020['Age group']=wca_train8020['Age group'].replace({'0': '0-35'})
      wca_eval8020['Age group']=wca_eval8020['Age group'].replace({'0': '0-35'})
```

FIGURE 5 TYPO FIXING

For variables *'Physiotherapy'* and *'Athlete Score'* we observed the existence of negative values, so we converted them to their absolute value.



```
[52]: wca_train['Athlete score'] = wca_train['Athlete score'].abs()
      wca_train7030['Athlete score'] = wca_train7030['Athlete score'].abs()
      wca_eval7030['Athlete score'] = wca_eval7030['Athlete score'].abs()
      wca_train8020['Athlete score'] = wca_train8020['Athlete score'].abs()
      wca_eval8020['Athlete score'] = wca_eval8020['Athlete score'].abs()
```

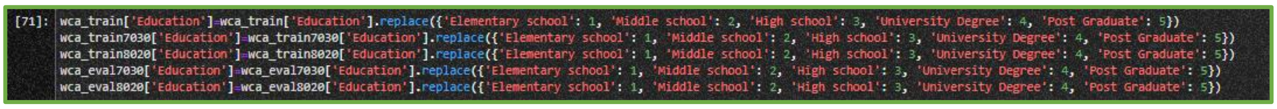
FIGURE 6 "FIXING" VALUES

Encoding non-numeric variables

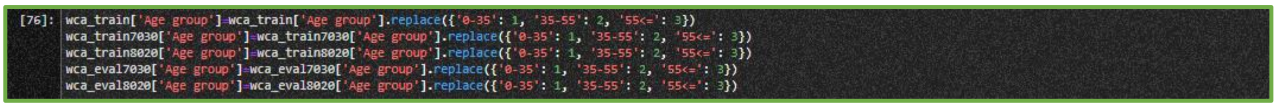
We also advanced to encode the non-numeric variables according to the qualities they describe. We used Ordinal Encoding on the ordinal non-numeric variables, and Binary Encoding on the variables that don't share this data type.

Variables	Strategy Employed	Reason
'Education', 'Age group', 'Income'	Ordinal Encoding	Possible Ordinal scaling
'Competition', '*Region'*	Binary Encoding	No logical order on the data sets' structure

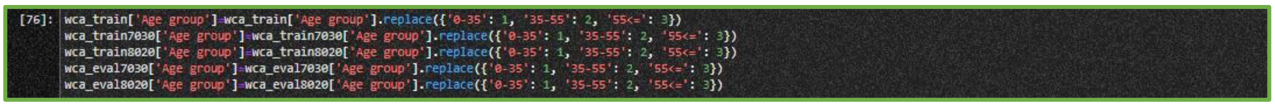
FIGURE 7 ENCODING SOLUTION



```
[71]: wca_train['Education']=wca_train['Education'].replace({'Elementary school': 1, 'Middle school': 2, 'High school': 3, 'University Degree': 4, 'Post Graduate': 5})
      wca_train7030['Education']=wca_train7030['Education'].replace({'Elementary school': 1, 'Middle school': 2, 'High school': 3, 'University Degree': 4, 'Post Graduate': 5})
      wca_train8020['Education']=wca_train8020['Education'].replace({'Elementary school': 1, 'Middle school': 2, 'High school': 3, 'University Degree': 4, 'Post Graduate': 5})
      wca_eval7030['Education']=wca_eval7030['Education'].replace({'Elementary school': 1, 'Middle school': 2, 'High school': 3, 'University Degree': 4, 'Post Graduate': 5})
      wca_eval8020['Education']=wca_eval8020['Education'].replace({'Elementary school': 1, 'Middle school': 2, 'High school': 3, 'University Degree': 4, 'Post Graduate': 5})
```



```
[76]: wca_train['Age group']=wca_train['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_train7030['Age group']=wca_train7030['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_train8020['Age group']=wca_train8020['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_eval7030['Age group']=wca_eval7030['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_eval8020['Age group']=wca_eval8020['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
```



```
[76]: wca_train['Age group']=wca_train['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_train7030['Age group']=wca_train7030['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_train8020['Age group']=wca_train8020['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_eval7030['Age group']=wca_eval7030['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
      wca_eval8020['Age group']=wca_eval8020['Age group'].replace({'0-35': 1, '35-55': 2, '55<=': 3})
```

FIGURE 9 ORDINAL ENCONDING OF VARIABLES EDUCATION, AGE GROUP AND INCOME

Data's First Statistics and Data Visualization

We took a brief first look at the variables and began to inquire on their possible impact on the athletes' outcome and the relationship between them.We were supported by the usage of scatter plots and correlation matrices.

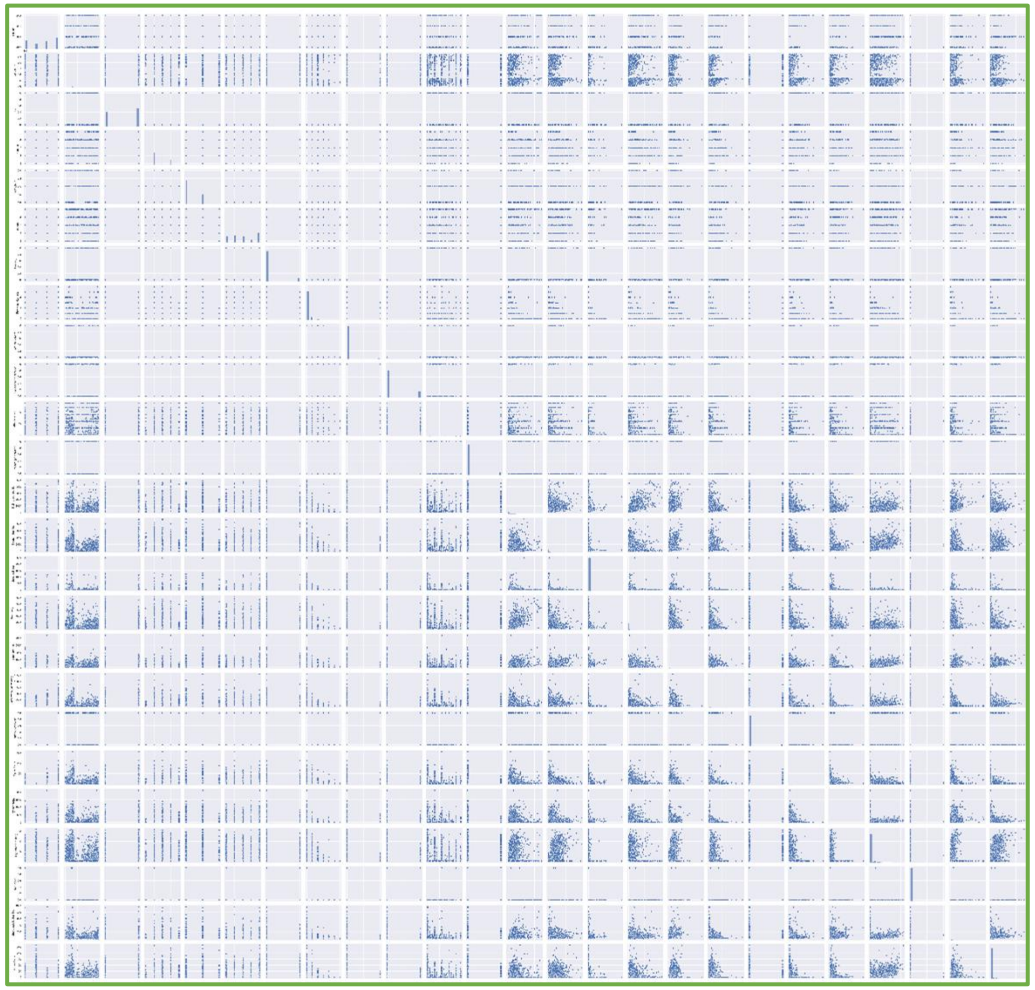


FIGURE 8 VARIABLES' SCATTER PLOT

Variables Correlation

We proceed to get a first glimpse on the variables' Pearson and Spearman's correlation among themselves.

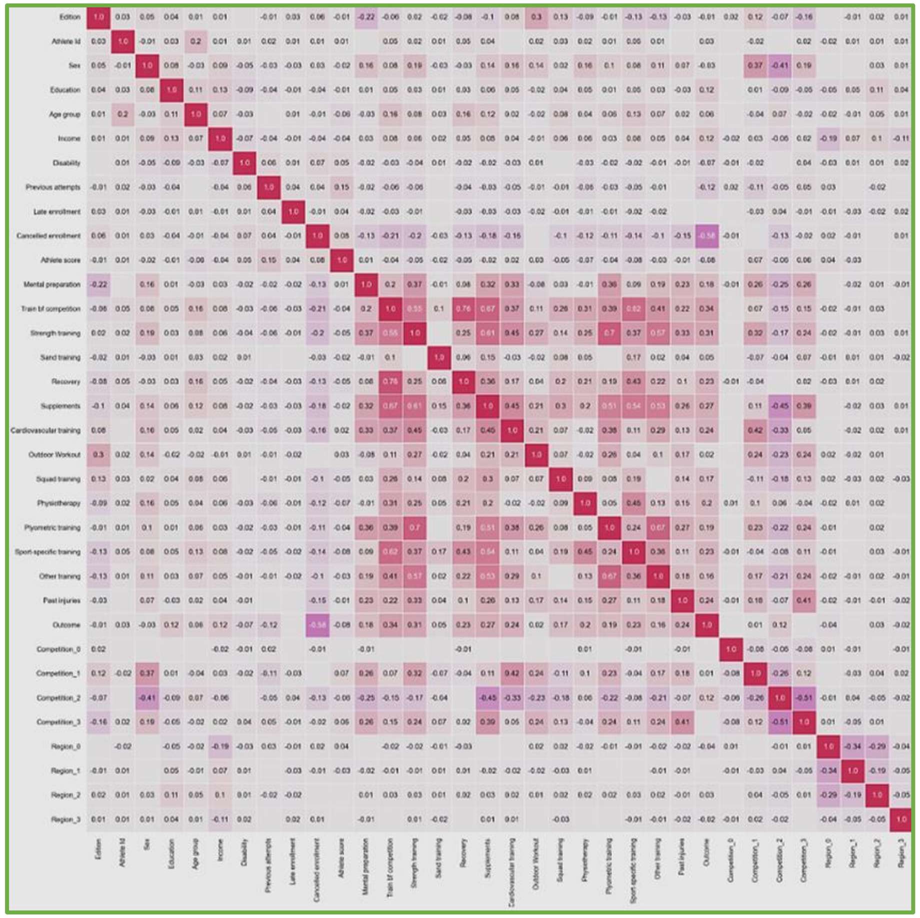


FIGURE 9 PEARSON CORRELATION MATRIX

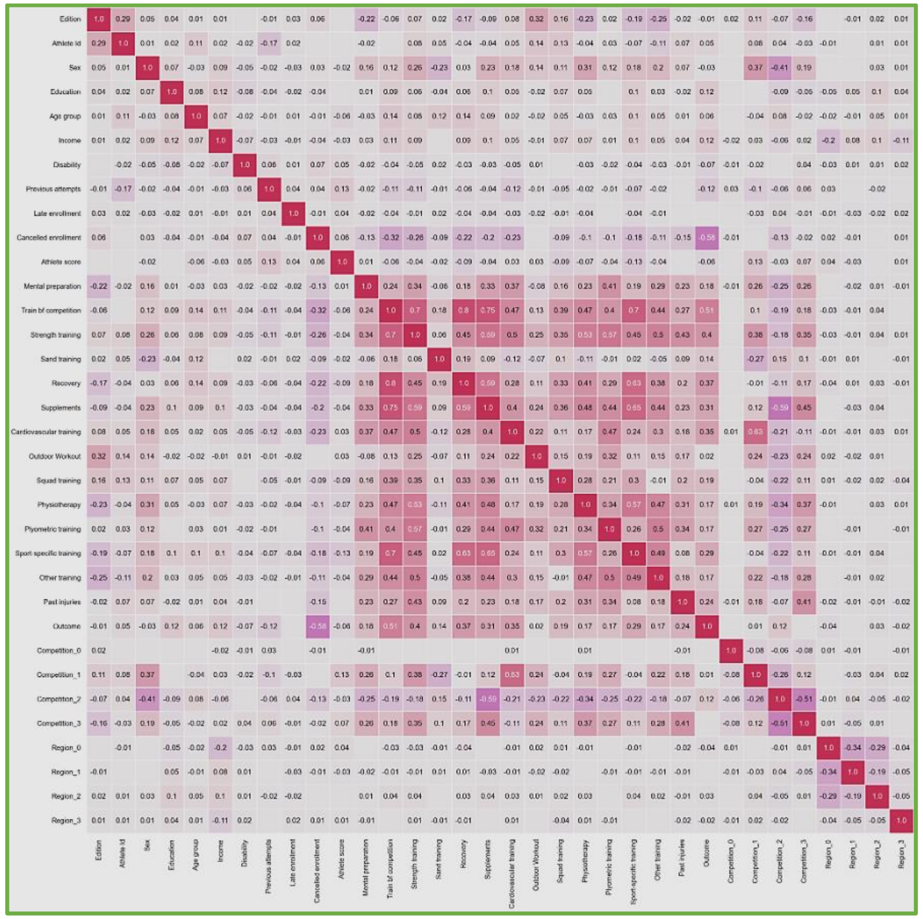
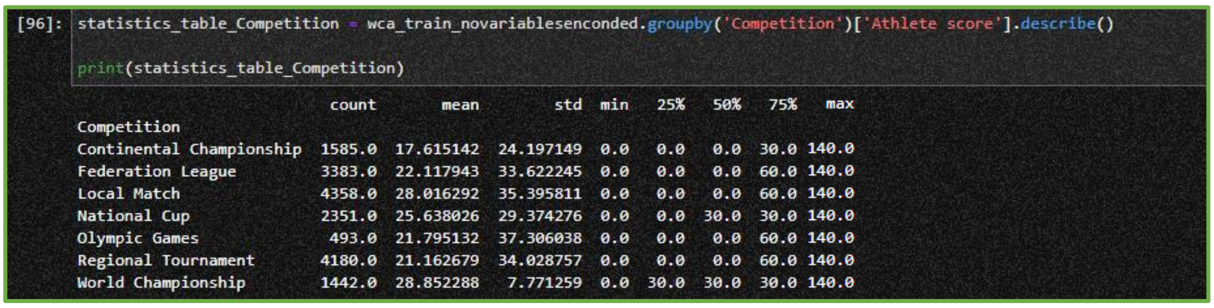


FIGURE 10 SPEARMAN CORRELATION MATRIX

Through the observation of both matrices, we were able to perceive certain type of relations that exist between the variables of the dataset.

As logically presumable, both variables *'Edition'* and *'Athlete Id* appeared to be correlated with their pairs. However, these correlations were not that strong. This made sense due to the already previously explored hypothesis that defends the existence of a heavy aleatory factor within sports competitions. For that reason, we would later consider on the necessity to include these two variables on our model prototype.

We could also attain on visioning the importance of the *'Competition'* itself on the athletes' performances as it is naturally expected that the biggest stages offer an extra stimulation (or a too much strong to handle pressure) comparing to the "smaller competitions". It is also important to remember that the quality of the athletes tends to increase with the competitions' quality: per example, for an athlete to participate in the Olympic Games he as to exceed in his "art".



```
[96]: statistics_table_Competition = wca_train_novariablesenconded.groupby('Competition')['Athlete score'].describe()

      print(statistics_table_Competition)
```

```
                           count       mean        std  min   25%   50%   75%    max
Competition
Continental Championship  1585.0  17.615142  24.197149  0.0   0.0   0.0  30.0  140.0
Federation League         3383.0  22.117943  33.622245  0.0   0.0   0.0  60.0  140.0
Local Match               4358.0  28.016292  35.395811  0.0   0.0   0.0  60.0  140.0
National Cup              2351.0  25.638026  29.374276  0.0   0.0  30.0  30.0  140.0
Olympic Games              493.0  21.795132  37.306038  0.0   0.0   0.0  60.0  140.0
Regional Tournament       4180.0  21.162679  34.028757  0.0   0.0   0.0  60.0  140.0
World Championship        1442.0  28.852288   7.771259  0.0  30.0  30.0  30.0  140.0
```

FIGURE 11 ATHLETE SCORE PER COMPETITION

We tried to cover the existence of any logical relation between the *'Age Group'* and the athletes' scores (*'Athlete score'*) but no substantial information was obtained.

Finally, we looked at the training type variables (variables ending with "training"). After checking on each variable's similar behaviour along the auxiliary target variable, *'Athlete score'*, we decided that we would be merging these variables to create a new variable, *'Training'*. However, as the general correlation differed between these variables, we weighted the sum of this variables based on the spearman correlation factor, advocating a $\frac{1}{correlation(trainingVar\,,\ Athlete\ score)}$ to each variable **(*)**. We normalize the weights to the percentage scale to better comprehend their impact. In the end this new variable would not be used as its inclusion on the model dataset did not improve the results.

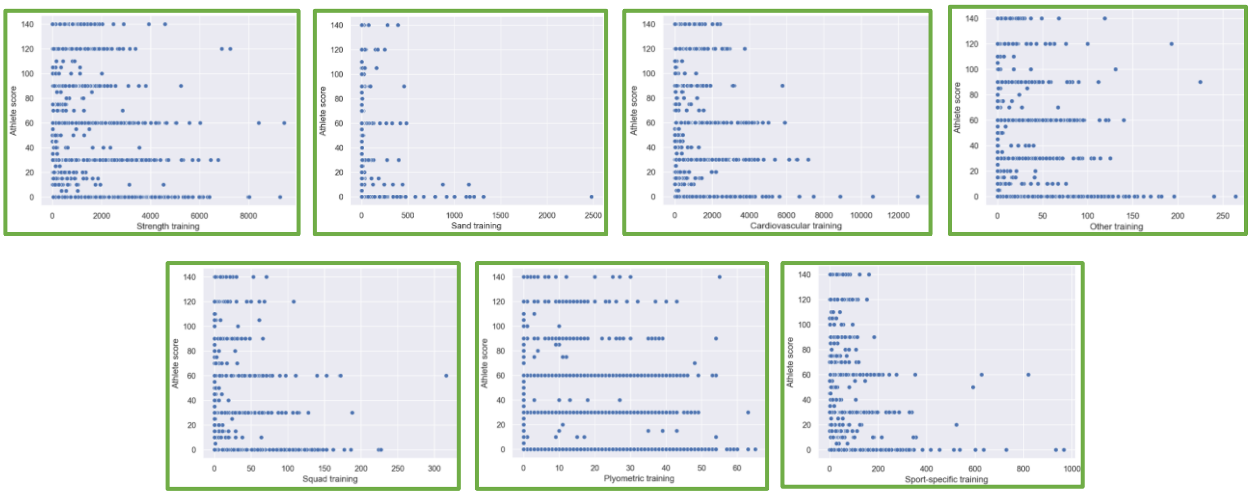


FIGURE 12 TRAINING VARIABLES VS ATHLETE SCORE

(*) Although we describe this process in this chapter, we only proceed with the creation of *'Training'* variable after using KNN Imputation

Data Pre-Processing

We were ready to finally start preparing the datasets for going through the modelling process. This step is of crucial importance to optimize the model global functioning and facilitates meaningful insights for decision-making and problem-solving.

Resolution on Atypical Non-Standard Values (Outliers)

In the process of removing the outliers we first tried to use both the IQR and standard deviation criteria to establish the boundaries between the values of each variable. However, the percentage of data that would be removed was too much for us to go forth with this techniques' overall usage. In the end we opted to drop outliers based on manually defined criteria for most of the variables. We were supported by the variables' histogram plots and their statistics.

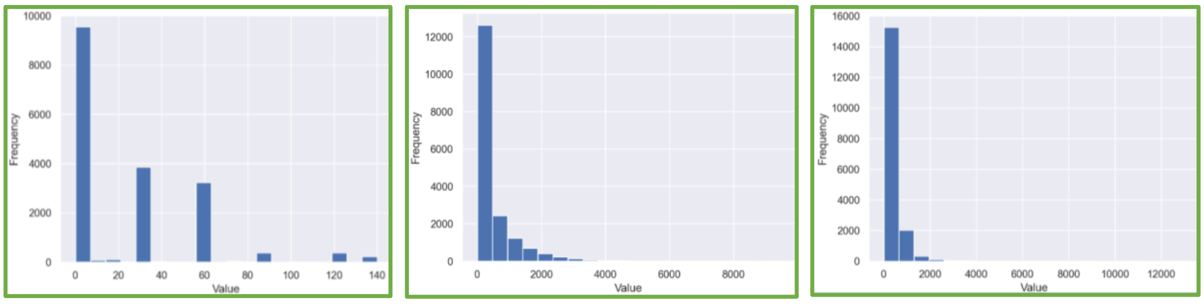


FIGURE 13 HISTPLOTS FOR VARIABLES ATHLETE SCORE, STRENGHT TRAINING AND CARDIOVASCULAR TRAINING

Below we find a resume on the outliers' removal criteria.

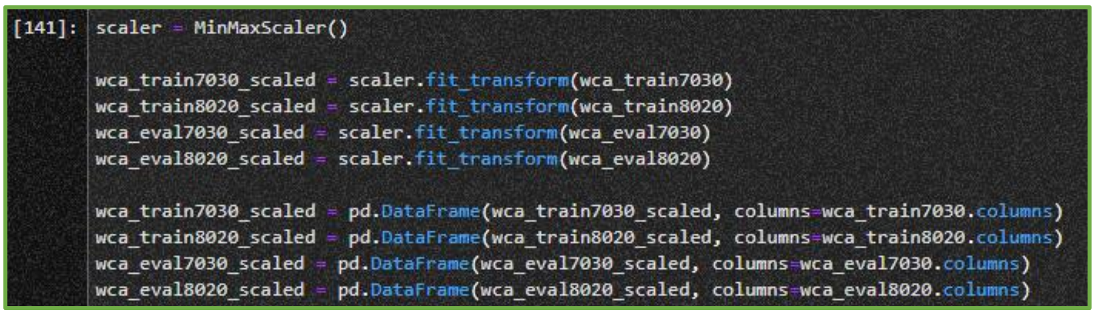
Variable	Droppage Criteria
'Athlete Score'	No dropped values
'Train bf competition'	Records with value >2500
'Strength training'	Records with value >4500
'Sand training'	Records with value >1500
'Recovery'	Records with value >6000
'Supplements'	Records with value >3000
'Cardiovascular training'	Records with value >7000
'Squad training'	No dropped values
'Physiotherapy'	No dropped values
'Plyometric training'	No dropped values
'Sport-specific training'	Records with value >900
'Other training'	Records with value >225

FIGURE 14 OUTLIERS DROPPAGE CRITERIA

After dropping these outliers, we ended up with <u>96.91%</u> of the 70/30 training dataset and <u>96.9%</u> of the 80/20 training dataset. Although we were aware of the instability still displayed within some of the variables caused by the existence of some non-standardized values, we would attenuate their effect with the data normalization.

Data Normalization

We decided to apply minmax scaling on both partitions. We decided on this as our variables structure was too much random and skewed. Also, by applying this scaling technique, we created an extra protection layer to reduce the outliers' impact, as the scaled values would be mapped within the minimum and maximum values of zero and one.

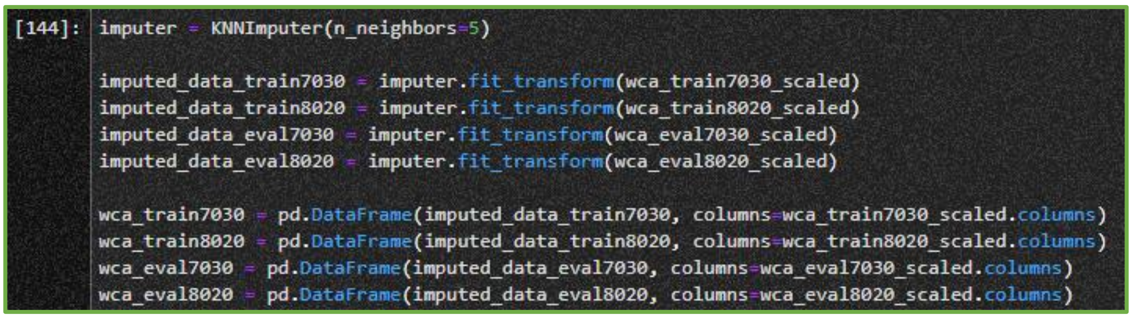


```
[141]: scaler = MinMaxScaler()

       wca_train7030_scaled = scaler.fit_transform(wca_train7030)
       wca_train8020_scaled = scaler.fit_transform(wca_train8020)
       wca_eval7030_scaled = scaler.fit_transform(wca_eval7030)
       wca_eval8020_scaled = scaler.fit_transform(wca_eval8020)

       wca_train7030_scaled = pd.DataFrame(wca_train7030_scaled, columns=wca_train7030.columns)
       wca_train8020_scaled = pd.DataFrame(wca_train8020_scaled, columns=wca_train8020.columns)
       wca_eval7030_scaled = pd.DataFrame(wca_eval7030_scaled, columns=wca_eval7030.columns)
       wca_eval8020_scaled = pd.DataFrame(wca_eval8020_scaled, columns=wca_eval8020.columns)
```

FIGURE 15 NORMALIZING THE SUB DATAFRAMES

KNN Imputation

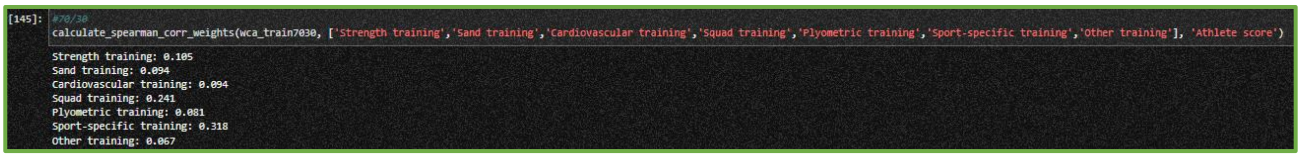
Finally, after disclosing on the outliers' removal and sub datasets normalization, we filled in the last of the remaining blank values using the predesigned function KNN Imputer (). As early mentioned, we decided on this approach was based on the diversity among the variables' distributions.



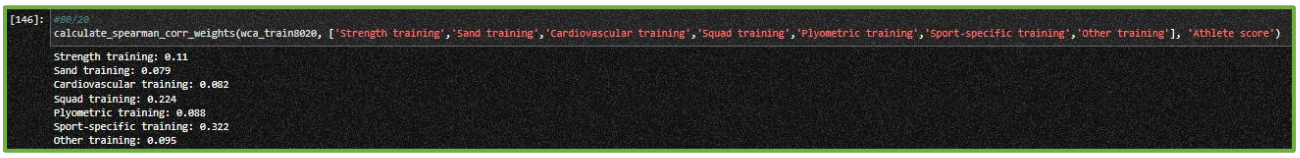
```
[144]: imputer = KNNImputer(n_neighbors=5)

       imputed_data_train7030 = imputer.fit_transform(wca_train7030_scaled)
       imputed_data_train8020 = imputer.fit_transform(wca_train8020_scaled)
       imputed_data_eval7030 = imputer.fit_transform(wca_eval7030_scaled)
       imputed_data_eval8020 = imputer.fit_transform(wca_eval8020_scaled)

       wca_train7030 = pd.DataFrame(imputed_data_train7030, columns=wca_train7030_scaled.columns)
       wca_train8020 = pd.DataFrame(imputed_data_train8020, columns=wca_train8020_scaled.columns)
       wca_eval7030 = pd.DataFrame(imputed_data_eval7030, columns=wca_eval7030_scaled.columns)
       wca_eval8020 = pd.DataFrame(imputed_data_eval8020, columns=wca_eval8020_scaled.columns)
```

FIGURE 16 KNN IMPUTATION ON REMAINING MISSING VALUES

After using the KNN Imputation we proceeded to create the new *'Training'* variable as we early introduced in previous chapter. The weights used in the weighted sum of variable *'Training'* within the 70/30 and the 80/20 partitions are displayed below.



```
[145]: #70/30
       calculate_spearman_corr_weights(wca_train7030, ['Strength training','Sand training','Cardiovascular training','Squad training','Plyometric training','Sport-specific training','Other training'], 'Athlete score')

       Strength training: 0.105
       Sand training: 0.094
       Cardiovascular training: 0.094
       Squad training: 0.241
       Plyometric training: 0.081
       Sport-specific training: 0.318
       Other training: 0.067
```



```
[146]: #80/20
       calculate_spearman_corr_weights(wca_train8020, ['Strength training','Sand training','Cardiovascular training','Squad training','Plyometric training','Sport-specific training','Other training'], 'Athlete score')

       Strength training: 0.11
       Sand training: 0.079
       Cardiovascular training: 0.082
       Squad training: 0.224
       Plyometric training: 0.088
       Sport-specific training: 0.322
       Other training: 0.095
```

FIGURE 17 WEIGHTS FOR 'TRAINING' VARIABLE WEIGHTED SUM CREATION

Second Data Understanding and Analysis

After assuring our dataframes were in a proper condition to be analysed, we got to use a motley set of statistical methods for achieving a final variable selection for our model assembly. We early decided that we would be dropping variables *‘Edition’*, *‘Competition’* and *‘Region_0’*.

Variables *‘Edition’* and *‘Competition’* would be dropped as we wanted to ensure the inter-independence of athletes’ trials. This approach allows for a clearer understanding of the athletes' abilities and performance without the potential influence or confounding effects of the specific edition of the event or the level of competition. Instead, it allows for the evaluation of the athletes' performance as an independent measure of their skill and capabilities.

The decision to drop variable *‘Region_0’* was intended to optimize the model structure as there were three more variables *‘Region_x’* that evaluate the same characteristic. By optimizing the model structure, unnecessary redundancy is reduced, which can help improve model performance and interpretability. Additionally, it can help mitigate the risk of multicollinearity, a statistical issue that arises when variables are highly correlated with each other, which can negatively impact model accuracy and stability.

At this point, after dropping variable *‘No coach’* and encoding the non-numeric ones, we had thirty-three variables.

Within the below board we find the different variable selection techniques and the respective selected variables after running each one of them. The results displayed are the ones for 80/20 partition.

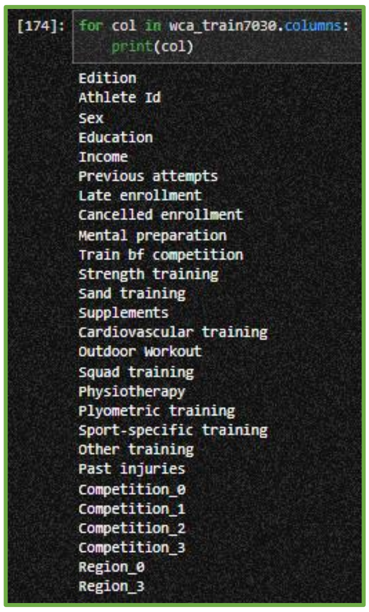
Method	**Selected Variables**
Wrapper Method **(*)**	*'Edition', 'Sex', 'Education', 'Income', 'Previous attempts', 'Cancelled enrollment', 'Mental preparation', 'Train bf competition', 'Supplements', 'Past injuries', 'Competition_0', 'Competition_1', 'Competition_2', 'Competition_3', 'Region_0', 'Region_3', ‘Training'*
Recursive Feature Elimination	*'Education', 'Previous attempts', 'Cancelled enrollment', 'Mental Preparation', 'Train bf competition', 'Past Injuries', 'Competition_0', 'Competition_2' 'Competition_3', ‘Training’*
Recursive Feature Elimination with Cross-Validation (cv=5) **(**)**	*'Edition', 'Sex', 'Education', 'Income', 'Previous attempts', 'Cancelled enrollment', 'Mental preparation', 'Train bf competition', 'Supplements', 'Past injuries', 'Competition_0', 'Competition_1', 'Competition_2', 'Competition_3', 'Region_0', 'Region_3', 'Training'*
Lasso Regression for Elimination **(***)**	*'Income', 'Cancelled enrollment', 'Train bf competition', 'Past injuries', ‘Training’*

FIGURE 18 VARIABLE SELECTIONS METHODS RESULTS

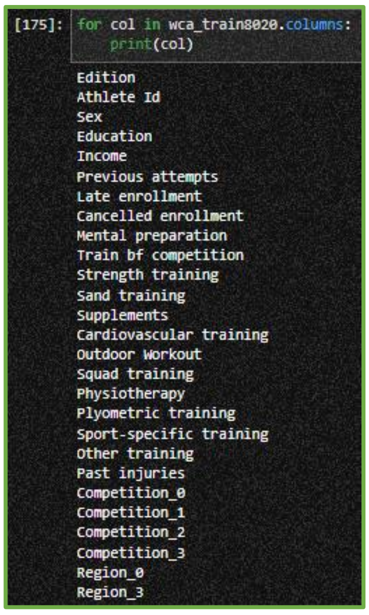
Both **(*)** and **(**)** techniques, when applied to the 70/30 partition dropped an extra variable (*‘Athlete Id’* and *‘Sex’*). Also, some variables changed from both selections when compared to the other partition, namely ‘Region_1’ instead of ‘Region_0’. In the case of the third technique, **(***)**, the 70/30 partition variables also feature *‘Competition_3’*.

In the end, we opted for using the <u>Recursive Feature Elimination with Cross-Validation (cv=5)</u> as the technique for variables selection on both subdivisions (70/30 and 80/20).

The usage of this technique aligned with the previous established "kick" of the three variables *'Edition'*, *'Competition'* and *'Region_0'*, led the dataframes to arrive the model assembly with the following variables' disposal.



```
[174]: for col in wca_train7030.columns:
           print(col)

       Edition
       Athlete Id
       Sex
       Education
       Income
       Previous attempts
       Late enrollment
       Cancelled enrollment
       Mental preparation
       Train bf competition
       Strength training
       Sand training
       Supplements
       Cardiovascular training
       Outdoor Workout
       Squad training
       Physiotherapy
       Plyometric training
       Sport-specific training
       Other training
       Past injuries
       Competition_0
       Competition_1
       Competition_2
       Competition_3
       Region_0
       Region_3
```



```
[175]: for col in wca_train8020.columns:
           print(col)

       Edition
       Athlete Id
       Sex
       Education
       Income
       Previous attempts
       Late enrollment
       Cancelled enrollment
       Mental preparation
       Train bf competition
       Strength training
       Sand training
       Supplements
       Cardiovascular training
       Outdoor Workout
       Squad training
       Physiotherapy
       Plyometric training
       Sport-specific training
       Other training
       Past injuries
       Competition_0
       Competition_1
       Competition_2
       Competition_3
       Region_0
       Region_3
```

FIGURE 19 FINAL VARIABLE SELECTION

Predicting the Outcome of an Athlete – Model Assembly and Evaluation

Pre-Assessing on the Prior Model Qualities

We prioritize evaluating models based on their F1-score, a critical metric that balances precision and recall. A high F1-score indicates a strong ability to accurately classify positive and negative instances. While F1-score is our primary focus, we also consider other important evaluation statistics and factors for a comprehensive assessment of the models.

Training Model Candidates

We tried eight (8) different types of algorithms to conceive our predictive model on the two different partitions of *WCA*'s modified train dataset: Linear Regression, Random Forest Classifier, Multi-Layer Perceptron Regressor, Balanced Random Forest Classifier, *RUSBoost* Classifier, Support Vector Machine and *XGBoost* Classifier.

We proceed to briefly explain the three last referred algorithms and why we decided to apply them.

Class imbalance is a common term to define the occurrence of an uneven number of values in the target variable, with one class significantly outnumbering the other(s). In this case our target variable *'Outcome'* is slightly imbalanced, with a light different on the number of "winners" (1) and "losers" (0) [70/30 portion with seven thousand and two hundred fifty-six (7256) winners and four thousand and four hundred thirty-one losers (4231) while the 80/20 has eight thousand two hundred ninety-five (8295) and five thousand six hundred thirty-five (5635) records of each value class. This can lead to biased models that favour the majority value class, resulting in a poor performance on the minority class.

Balanced Random Forest Classifier is and adaptation of Random Forest Classifier where the issue of class imbalance is addressed by adapting the sampling strategy during the construction of individual decision trees. This helps in reducing the bias towards the majority class and improving the predictive performance for minority classes.

RUSBoost Classifier (**Random Under-Sampling Boosting Classifier**) aims to mitigate this issue by reducing the imbalance through random under-sampling and improving classification accuracy through boosting. Then, *RUSBoost* trains decision trees on a balanced training set and assigns higher weights to misclassified instances. This process is repeated to create an ensemble of weak learners. Predictions from the weak learners are combined using techniques like weighted voting or averaging. After, *RUSBoost* adjusts the data distribution to improve predictive performance on both classes.

SVM (**Support Vector Machine**) is a classification algorithm used for predictive model design. It is also its effectiveness in handling imbalanced datasets that led us to apply this modelling process. SVM finds an optimal decision boundary by maximizing the margin between classes. It is supported by support vectors (instantiations that lie closest to the decision boundaries) that agglomerate the similar points in the hyperplanes on the variables space. SVM can adjust class weights, incorporate oversampling or under sampling techniques, and even perform one-class SVM for extreme imbalances.

XGBoost Classifier (**Extreme Gradient Boosting Classifier**) is a highly effective classifier algorithm used in predictive model design. It also stands out for its ability to handle imbalanced datasets in a versatile way. It works by iteratively combining weak models, typically decision trees, to create a robust and accurate ensemble classifier. This boosting process focuses on correcting errors made by previous models, allowing *XGBoost* to learn intricate patterns within the data and make precise predictions. By adjusting the weights assigned to the data instances during the boosting process, *XGBoost* emphasizes the importance of minority class samples, enabling the algorithm to learn from them more attentively. For that reason, this model incorporates an extra powerful regularization mechanism, which helps prevent overfitting and enhances generalization.

Model Evaluation

In the table below we find the evaluation measures on each one of the implemented prediction models.

Model	Portion (Train / Eval)	Parameters	F1 Score	Other Measures (Evaluation)
Logistic Regression	70/30	penalties according to cost matrix = {0: 1, 1: 10}	Train: 0.8439 **//** Evaluation: 0.8493	Sensitivity: 0.9997 // Specificity: 0.4747
	80/20		Train: 0.8534 **//** Evaluation: 0.8502	Sensitivity: 0.9994 // Specificity: 0.4825
Random Forest Classifier	70/30	n_estimators: 500 min_samples_split: 20 min_samples_leaf: 20 max_features': 'log2' 'max_depth': 10 'bootstrap': True	Train:0.8981 **//** Evaluation: 0.8965	Sensitivity: 0.9174 // Specificity: 0.8085
	80/20		Train: 0.8979 **//** Evaluation: 0.8949	Sensitivity: 0.9155 // Specificity: 0.8065
Balanced Random Forest Classifier	70/30		Train: 0.8978 **//** Evaluation: 0.8974	Sensitivity: 0.9186 // Specificity: 0.8094
	80/20		Train: 0.8985 **//** Evaluation: 0.8978	Sensitivity: 0.9202 // Specificity: 0.8079

***RUSBoost* Classifier**	70/30	n_estimators=50
		random_state=45
			Train: 0.8759 **//** Evaluation: 0.8820	Sensitivity: 0.8990 // Specificity: 0.7933

	80/20		Train: 0.8842 **//** Evaluation: 0.8895	Sensitivity: 0.9127 // Specificity: 0.7934
SVM	70/30	'C': 10 'gamma': 'scale' 'kernel': 'rbf'	Train: 0.8948 **//** Evaluation: 0.8854	Sensitivity: 0.9416 // Specificity: 0.7251
	80/20		Train: 0.8940 **//** Evaluation: 0.8879	Sensitivity: 0.9552 // Specificity: 0.7091
K-Nearest Neighbours Classifier	70/30	‘n_neighbors’: 15 ‘p’: 1 ‘weights’: ‘distance’	Train: 1.00 **(*)** **//** Evaluation: 0.8667	Sensitivity: 0.9295 // Specificity: 0.6800
	80/20		Train: 1.00 **(*)** **//** Evaluation: 0.8709	Sensitivity: 0.9365 // Specificity: 0.6890
***XGBoost* Classifier**	70/30	'subsample': 1.0 'reg_lambda': 1.0 'reg_alpha': 0.4 'n_estimators': 100 'min_child_weight': 1 'max_depth': 5 'learning_rate': 0.07 'gamma': 0 'colsample_bytree': 0.9	Train: 0.9112 **//** Evaluation: 0.9090	Sensitivity: 0.9605 // Specificity: 0.7735
	80/20	'subsample': 1.0 'reg_lambda': 0.1 'reg_alpha': 0.2 'n_estimators': 215 'min_child_weight': 5 'max_depth': 5 'learning_rate': 0.1 'gamma': 0 'colsample_bytree': 0.9	Train: 0.9263 **//** Evaluation: 0.9121	Sensitivity: 0.9585 // Specificity: 0.7878
Multi-Layer Perceptron Regressor	70/30	‘solver’: ‘adam’ ‘max_iter’: 1000 ‘learning_rate’: ‘adaptive’ ‘hidden_layer_sizes’: (100,100) ‘alpha’: 0.01 ‘activation’: ‘tanh’	Train: 0.90358 **//** Evaluation: 0.90571	Sensitivity: **0.9608//** Specificity: 0.7615
	80/20		Train: 0.9039 **//** Evaluation: 0.9045	Sensitivity: 0.9505**//** Specificity: 0.7760

FIGURE 20 MODELS’ EVALUATION MEASURES

(*) The values for the parameters that are displayed above are those which let us have simultaneously the best results on both training and evaluation partitions, trying to avoid overfitting. However, we could not avert this effect using the K-Nearest Neighbours Classifier algorithm. We will explore some ideas that may be causing this effect.

We didn't select the *XGBoost* Classifier, despite its good F1 score. Instead, we chose algorithms from the scikit-learn library. Scikit-learn is a widely trusted and established machine learning library, known for its optimized implementations and strong community support. The library has undergone extensive validation and testing, ensuring reliable performance. It provides a comprehensive set of tools and algorithms that are widely used in the field.

As it was earlier mentioned, the KNN Classifier algorithm was overfitting regardless of chances of tuning it. This algorithm is prone to overfitting due to its non-parametric nature and fixed number of neighbours used for classification: the algorithm does not make assumptions about the data distribution, making it susceptible to memorizing the training data too closely. Parameter tuning has limited impact on addressing overfitting as KNN's hyperparameters, such as the number of neighbours, cannot effectively control the decision boundary complexity. The consistent overfitting of KNN, along with the relatively good performance of other models, suggests that KNN's reliance on local neighbourhood information contributes to its overfitting tendency.

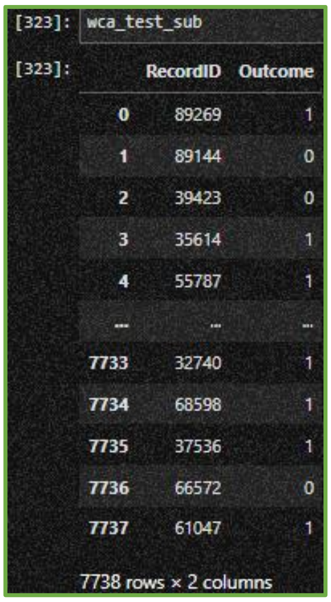
In the end, we opted to use the Multi-Layer Perceptron (MLP) algorithm (trained with the 80/20 partition sub dataframes) to establish our model prototype.

Evaluation Dataset Measures	**f1-score**	**accuracy**	**sensitivity**	**specificity**	**AUC**
MLP 80/20	0.9045	0.8802	0.9505	0.7760	0.8633

We went with this decision as this model was the one with most solid evaluation measures, especially the f1 score.

Prediction on *test* dataset

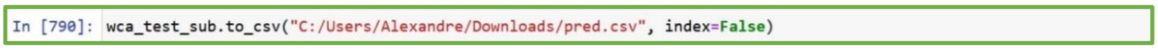
After applying the same coherence filtering to the *test* dataset as the one used for the model assembly, encoding the non-numeric variables and normalizing the whole dataset, we proceeded to run our model prototype resulting on a series of *'Outcome'* predictions.



```
[323]: wca_test_sub

[323]:       RecordID  Outcome
          0     89269        1
          1     89144        0
          2     39423        0
          3     35614        1
          4     55787        1
        ...       ...      ...
       7733     32740        1
       7734     68598        1
       7735     37536        1
       7736     66572        0
       7737     61047        1

      7738 rows × 2 columns
```

FIGURE 21 PREDICTIONS ON TEST DATASET

After this we would export our wca_test_sub dataframes, creating external csv files, which would be the ones used for the competition submissions within Kaggle.

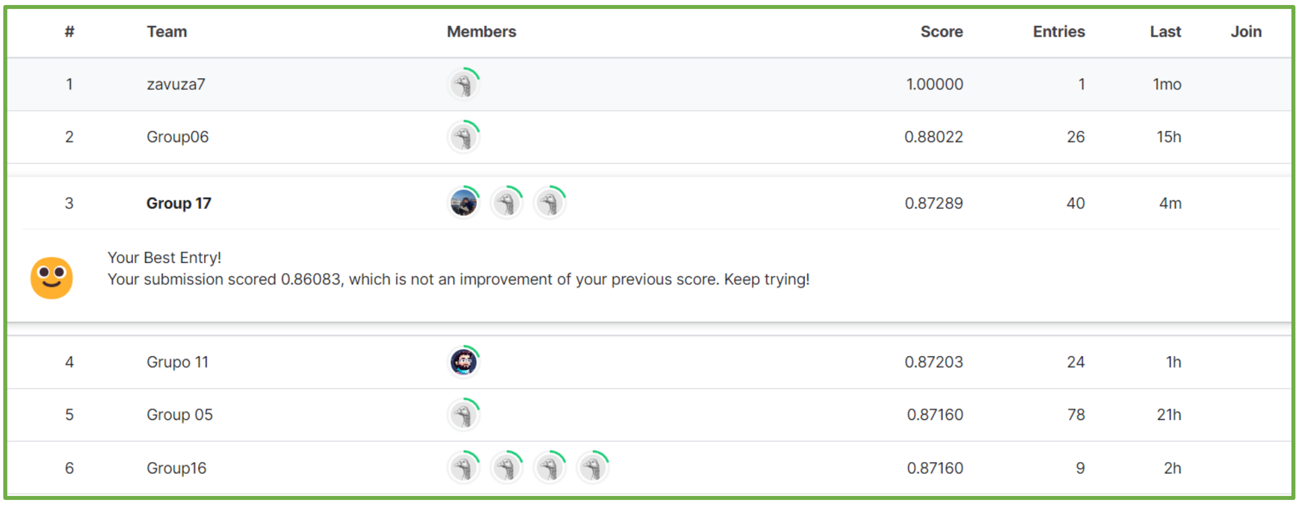


```
In [790]: wca_test_sub.to_csv("C:/Users/Alexandre/Downloads/pred.csv", index=False)
```

FIGURE 22 EXPORTING PREDICTIONS ON TEST DATASET TO EXTERNAL CSV

Kaggle submission and Conclusion

By the time of our last submission on the Kaggle competition, our group add achieved a f1-score of 0.87289 that ranked us on the second (2nd) place at that time.



#	Team	Members	Score	Entries	Last	Join
1	zavuza7		1.00000	1	1mo	
2	Group06		0.88022	26	15h	
3	**Group 17**		0.87289	40	4m	
4	Grupo 11		0.87203	24	1h	
5	Group 05		0.87160	78	21h	
6	Group16		0.87160	9	2h	

Your Best Entry!
Your submission scored 0.86083, which is not an improvement of your previous score. Keep trying!

FIGURE 23 LEADERBOARD AT TIME OF LAST SUBMISSION

It's important that due to the fickleness of the assembly process of our partition sub datasets, the results may slightly differ to the ones we obtained. This is mainly due to the proportion of "big" / "small" values that are within the training and evaluation dataframes, respectively.

After achieving our prototype, we were confident on our approach. On the other hand, we discuss some topics where we think we could have tested different solutions for a richer range of options.

1) Try to include our outliers in the model assembly, either by using their training power for prevent overfitting or just to have a simple line of comparison between the models that were ran with and without them.

2) Try to use a more extensive range of encoding options for the non-numeric variables.

3) As we did with the *'Training'* variables, we could have tried to increase our model compactness with an even more developed feature engineering. Per example, we could have tried to merge variables related with injuries and recovery time of an athlete.

4) Use a more diverse spectrum of hyperparameters within our models, not getting too much acquainted with the ones outputted via *RandomizedSearch* and *GridSearch*. We could have also tried to implement one or two more model algorithms.