Group 1 08.03.2021

Project Report Scalable Data Storage

**Intro**

As a group, we quickly found a topic which involved a passion of every one of us, the National Basketball Association (NBA). We wanted to deep-dive on the possibilities and capabilities of the techniques that we saw in this course and the analysis and predictions we could achieve. To reach this goal, we firstly needed to gain the data. All the Information about Players, Teams and Games were published by the NBA on their official webpage, [www.NBA/stats.com](http://www.NBA/stats.com). This collection reaches back until the year of 1980 bearing the data about every player since then until today and every game that has been played.

We thought about a scenario in which data and statistics about Players and Games can be used to predict the outcome of matches and give a more profound Analysis about behaviours of both. We were hoping to correctly predict if the Home team would win in the scheduled Matches occurring in the NBA.

The problem definition for this project could formulated by the objective of analysing NBA statistics of every game and team over a specific season.

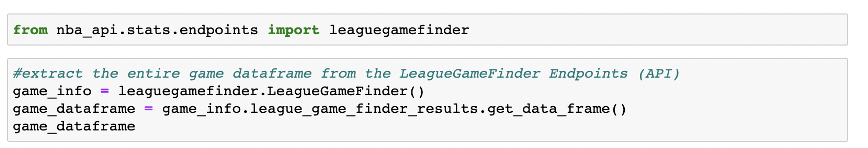
The solution we are presenting today consists of two major components. First, our NoSQL Database and second the Spark environment and the prediction Model that we established. We decided to use MongoDB as our NoSQL DBMS and initialised a cluster to which we uploaded the data for further configuration. We were fortunate to find nba\_APi an API Client for the NBA website which provided us with the Data. This package is meant to make the API Endpoints more accessible and to provide extensive documentation about the gathered Data. The usage of MongoDB allows us to store and perform any configuration needed on the data, which guarantees us the quality and a first configuration layer of the processed data.

In a second step we set up a connection between our MongoDB cluster and Spark. To enable the connection, we had to run the MongoDB Connector for specific functions and implicits for the SparkContext and RDD. Connection to MongoDB happens automatically when an RDD action requires a read from MongoDB or a write to MongoDB. We used the Spark environment to analyse the Data in depth and predict the outcome of future games. To achieve a prediction which is valuable in its result, we were confronted with the problem which variables to use. Here we decided to create a new variable, called ELO, which helps us to predict the performance of a Team based in their previous performances. Using the model, we have created gives us the opportunity to predict if a Hometeam will win or lose their next match. Furthermore, we also predict how offensive a Team will supposedly play in the next games by using the data about offensive and defensive Rebounds.

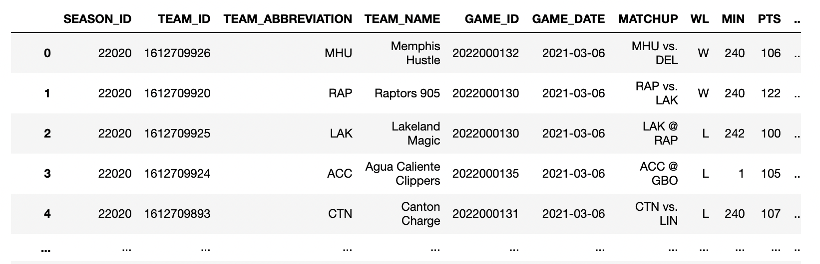
Our Motivation was to experience how well we could predict real events and how well we could navigate thru this big dataset.

The api allows us to make some interesting queries that extract all the information we need in a convenient way; thanks to the fact that is directly connected to the official NBA website that update data day by day.

Our entire work is built on a dataframe extracted from an endpoint package called “leaguegamefinder”. This package allows the extraction of all the games played and the related statistics.



The following dataframe was our starting point, made up of 30,000 rows and 28 columns.



Each row is an NBA game, from the 2014 season to the present with the relative statistics.

This are some of the most important stats that we decided to exploit:

|  |
| --- |
| FG\_PCT: Field Goal Percentage: The percentage of field goal attempts that a team makes |
| FG3\_PCT: Three points Field Goal Percentage |
| REB: This statistic is the number of total rebounds a player or team has collected on either offense or defense DREB, OREB: the n. of defensive rebounds and offensive ones |
| AST: The number of assists |
| PF: The number of fouls a team committed |

**NoSQL**

We decided to adopt the use of a NoSQL database, more specifically we used **MongoDB**. We chose MongoDB DBMS mainly for these reasons: it is open source, it is highly documented and is very flexible about the shape of individual documents within a collection.   
MongoDB is a *document-oriented database*, within it we can find data divided into collections and each record inside it is represented as a document. The documents are stored as a *JSON file*, more precisely in MongoDB are store as a BSON.

With MongoDb we were able to import the data into a *cluster*, where we have a database storing collections of documents that contain the information. In order to connect to this cluster with python and being able to query on it we used the *MongoClient* and **Pymongo**.

We worked with the cloud and free version of MongoDB, **Atlas**, which turned out to be perfect for the creation of the database. After a quick setup of the environment, we create the direct connections between MongoDB and our application (in this case a Jupyter Notebook). We have also created the link with **Mongo Shell**, having previously installed MongoDB on our machine; finally, we have connected MongoDB **Compass.** MongoDB Compass is the GUI client for MongoDB that enables you to view the DB and to create some queries.

We then created into MongoDB the collections of interest and imported the documents as json files, the json files which had been appropriately fetched and merged in advance using python.

We decided to use Mongo Compass, which is an application made for using remotely but needs to connect to the cluster created on Atlas. From there we imported the data and collections.

• Motivation of the choices above

The motivation behind the decision of using mongoDB is because we were working on Json files, which are a document based way to store information and MongoDB is build to work on this way of storing data.

Also, the connection between python and mongo was useful and better for us in order to query results and retrieve information from the API and Mongo. Mongo provides Mongo Compass, that as mentioned before, we used to upload the data and have a better data visualization quality of each collection.

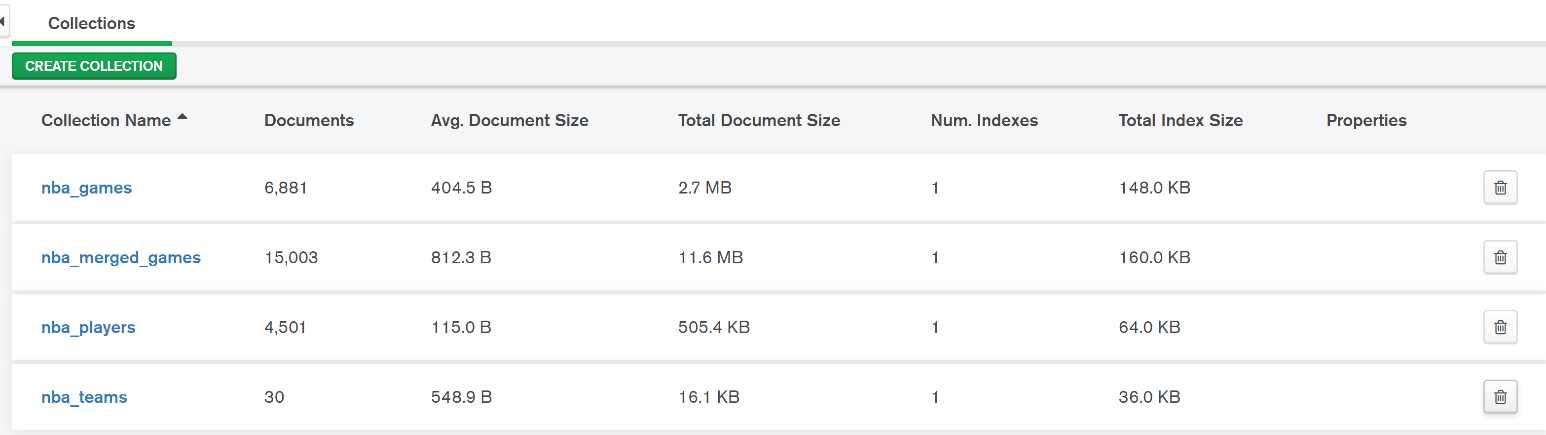
Finally, we decided to use these collections in order to have better data to analyze, knowing we need a solid database that can give us the information needed to do extensive analysis and using these collections of documents.

**Description of the Proof of concept**

From that, we imported each collection in the database. The collections used in the database are “nba\_players”, “nba\_teams” and “nba\_games”.



Each collection is unique and contains the information needed to perform an extensive analysis of the teams, players and games of the NBA. Is important to remember that a collection is storing various documents (json files) containing information related to what we need.

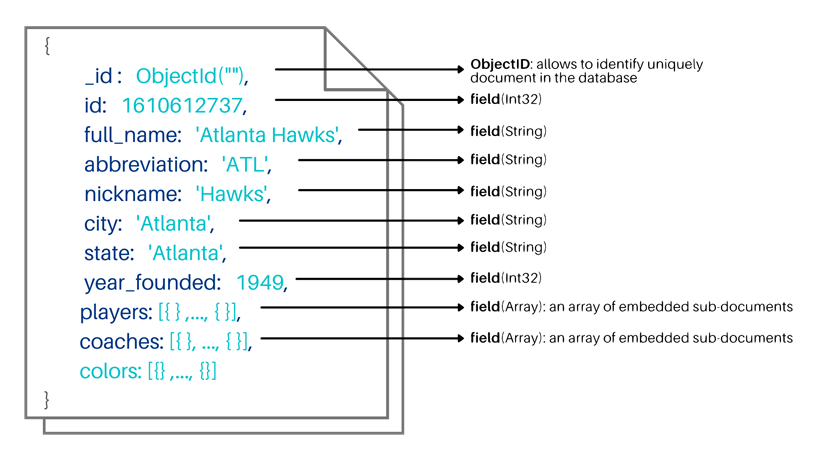
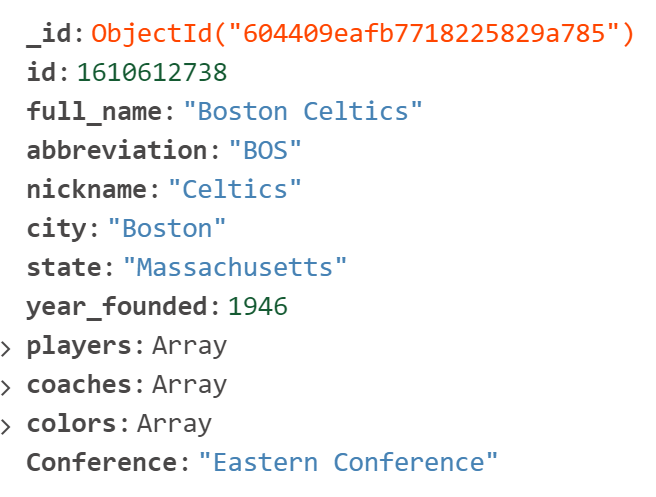


* This screenshot shows the collections of documents and how many documents are present. Furthermore we gain more insight about the structure of the data.

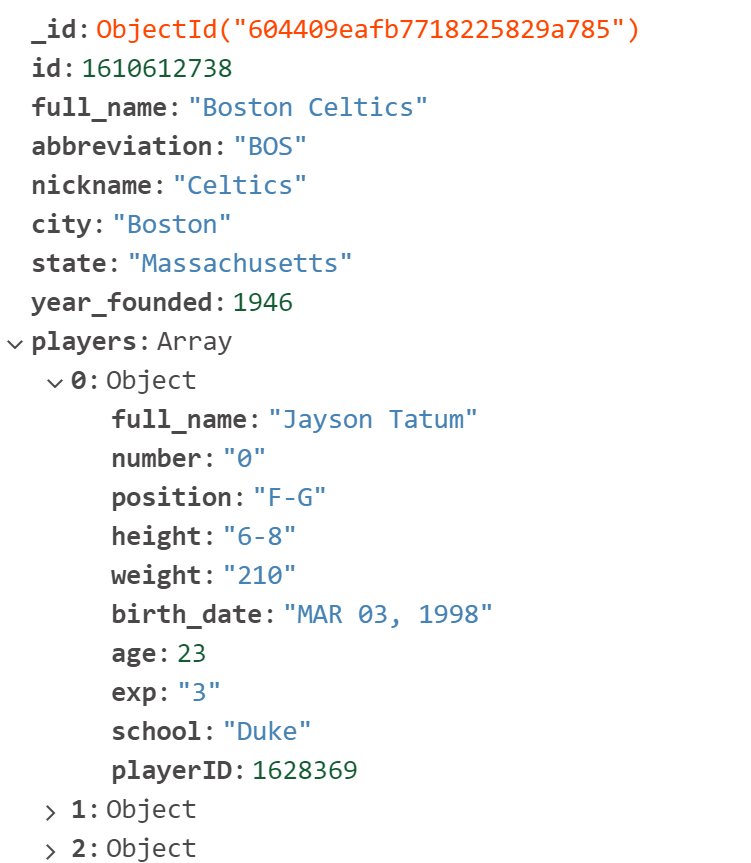
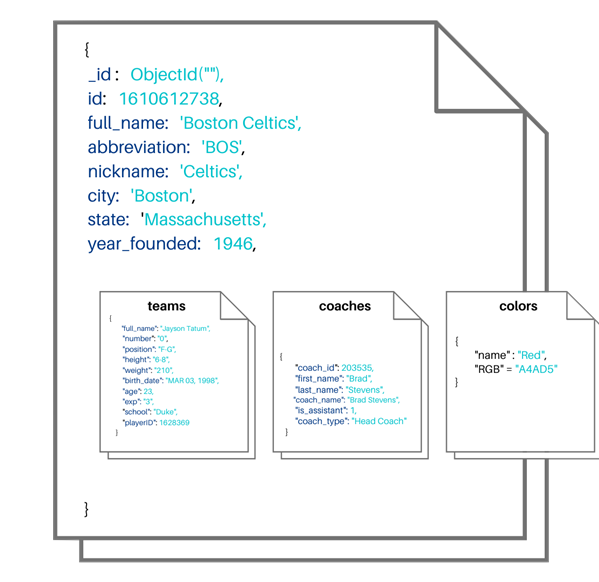
In the picture we can see nba\_merged\_games, which is a collection that is combining games in one line, for this we downloaded more documents and is showing more games in just one document insted of a game with two documents as it is in the nba\_games. Since we have a document for the home team and one document for the away team, in merged games we have both in the same document and their stats per game.

**NBA Teams Collections**

For purposes of the project we decided to go more into detail with the nba\_team collection since is the more rich document because we have inside various embed documents and we used it more for the queries.

This is an example of how the nba\_team is structured and on the right we see the real DB in mongo. We can see that there are elements that are strings, objects, arrays int32 and strings. Then we can see the embed documents in the next picture and how it looks in mongo db.



* Here we can see a basic structure of the teams collection, where we can see that we have embed documents in the collection of teams in the nba for players and coaches that would be objects.
* Is important to mention that for the Conference, we didn’t had it on the database, and we used the “update\_many” operator in order to implement this into the database.

Structure db, collections, hierarchy, teams and players. For teams the full document and another with all the arrays opened.

**CRUD Operations:**

We applied some *update operations* on our nba\_teams , modifying the existing documents in this collection. We have created a new conference field and using the $in operator we specify to which conference the team belongs, based on the value of the “full\_name” field.

In this operation, we decided to filter the teams that where founded in a specific period of time and are not from the state of California. For this we used:

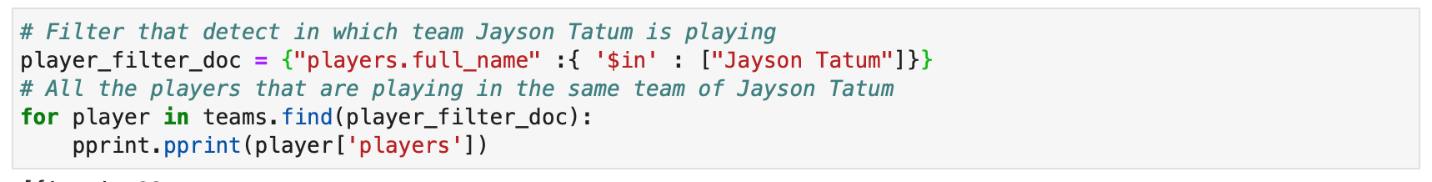
1. $gte, which is a operation that compares logically greater or equeal to, in this case we are searching for year founded greater or equal to 1940.
2. $lte which is the operator comparing logically less or equal, which means less or equal to. We can see that we completed the filter meaning that we need teams founded between 1940 and 1955 and are not from california.



For this filter we decided to search for the teams that have specific color on their logos. For this, we used the $elemMatch operator. This operator will search a match between the documents and subdocuments that have at least one element matched to the criteria we used. The difference between this operator and a regular match is that $elemMatch talkes into account the sub-documents in a document. In this case, is the color “red”.



In this filter, we are searching for players that are teammates of Jayson Tatum. For this we decided to use a filter on players and applying $in for the name of Jayson Tatum. From that, we create a for loop in order to retrieve the information of the players in the document that play with Jayson Tatum.



**Running some aggregations**

*MongoDB aggregation framework* is modeled on the concept of data processing pipelines, the function aggregate takes in a list of aggregation pipeline stages. The main idea is that we take as input a MongoDB collection and pass that collection through all the different stages, each of performs a different operation on its input.   
  
We started with a simple sanity check, computing the first 4 teams ever founded in NBA history. This can be achieved using a 3-stage pipeline:

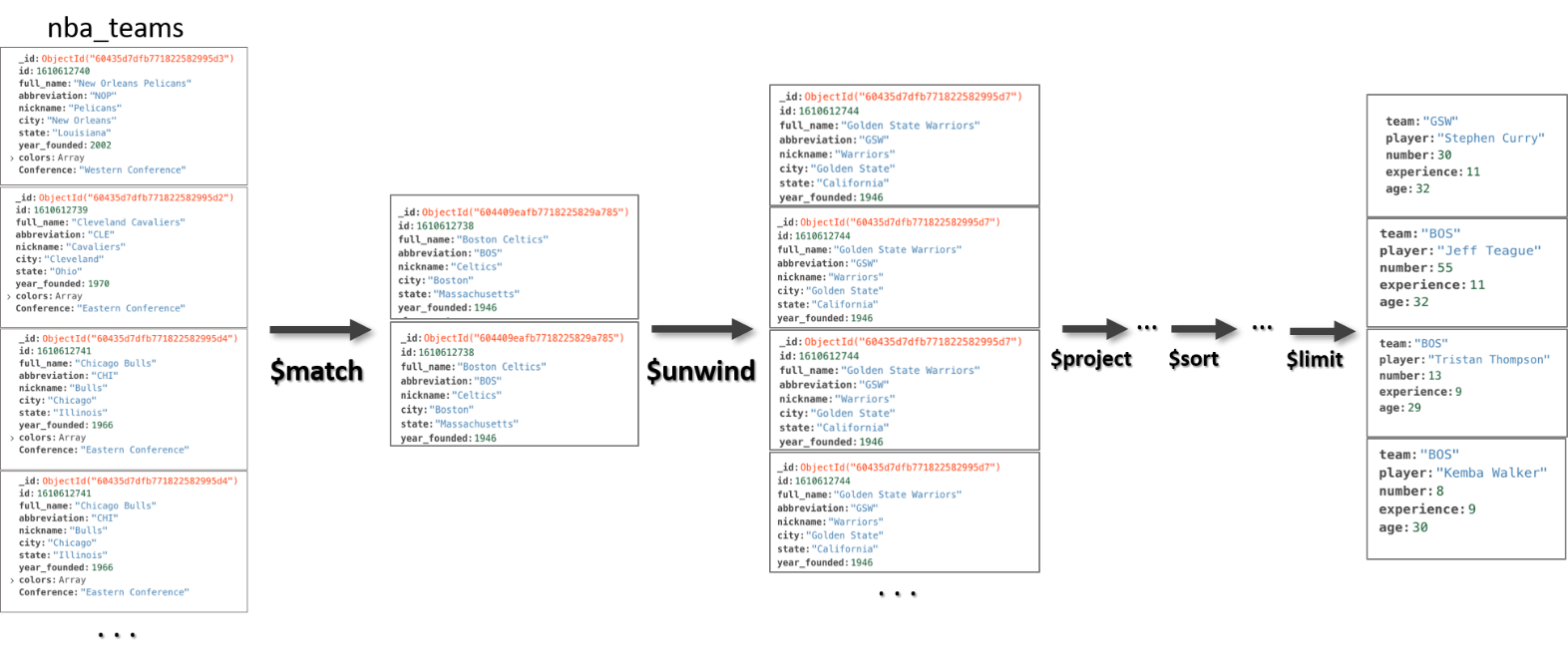
1. Use the $project stage to specify which fields to include.
2. Use the $sort stage to sort by the year of foundation, ascending.
3. Use the $limit stage to limit to the first 4 team founded



Describe aggregation

1. Use the $match stage to limit the teams that play in Golden State city
2. Use the $unwind stage to generate one document for each player in the team
3. Use the $project stage to specify which fields to include
4. Use the $sort stage to sort by the experience of the player, descending.
5. Use the $limit stage to limit to the 5 most experienced players

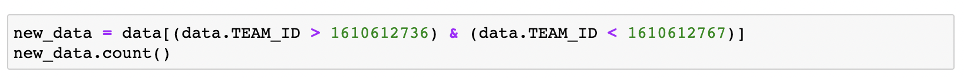




**CLEANING AND DATAFRAME PREPARATION**

Before moving on to the execution of queries or models it was necessary to clean the dataframe and to organize it in the most qualitative way possible.

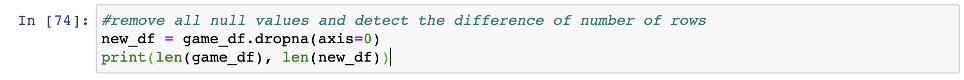
First of all we removed the games that referred to the lower leagues (NBA G League). In order to consider only NBA games. To do this we made a filter based on the id of the NBA teams.



Our dataframe stores all the games grouped according to a single team so for each match in the dataset there are two different lines, one with the statistics of the home team and the other with the guests one. We decided to merge the game id in order to have both teams on the same line, a necessary step for the construction of our predictive model.

We create a “combine\_team\_games” function that joined every row to all the others with the same game ID, filtering out the row that are joined to itself and modifying the previous columns name, specifying the \_home and \_away suffixes. After that, we proceed with a simple work of cleaning.

We searched if there is the presence of duplicates and removed the null values with the “dropna” pandas function.



**ELO FEATURE BUILDING**

The ELO rating represents the best existing method to quantify an NBA team's strength and performance over many seasons. All teams start at a median score of 1500 and are given, or subtracted points based on the final score of each game and where it was played with weights given to the margin of victory, location and to the ELO of the opposite team. It is a sort of win-loss record but that take in consideration more other stats.

This is the ELO rating formula:



S\_team: state variable 1\0 if the team wins or loses.

E\_tema: expected win probability of the team

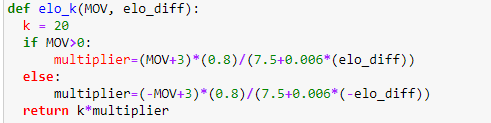
K: moving constant that depend on the margin of victory and difference in ELO rating between the two teams that are playing the match.

Elo ratings carry over from season to season and if R represent the final ELO of a team in one season, it's ELO rating at the beginning of the next season is approximately:

(R\*0.75)+(0.25\*1505)

We calculated it by using different kind of functions that take in consideration factors like for example: the win probability, the home curt factor and the actual level of skills and keeping it updated after every match.

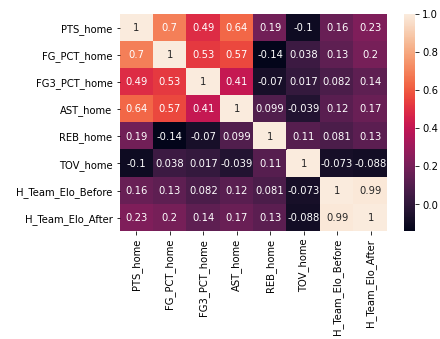
This function for example determines the constant used in the Elo rating, based on margin of victory and difference in Elo ratings:



we can get key insights about the strength of teams throughout seasons.

**CORRELATION MATRIX FEATURES**

In the end, in order to choose which features are the best for our predictive model we build a correlation matrix using the heatmap to visualize it better.



On the base of that, the features that we are going to choose for feeding our model are the followings:

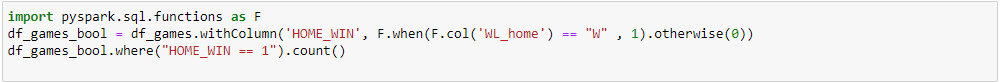
|  |  |
| --- | --- |
| FG\_PCT | Representing the accuracy of the team as field goals made divided by the number of attempts |
| FG3\_PCT | Representing the accuracy of the 3 points attempts |
| OREB | Representing the rebounds made by a team during an offensive action |
| DREB | Representing the rebounds made by a team during a defensive action |
| AST | Representing the assist made |
| Team\_Elo\_Before | Representing the ELO rating before that match |
| Team\_Elo\_After | Representing the ELO rating after that match |

**PySpark**

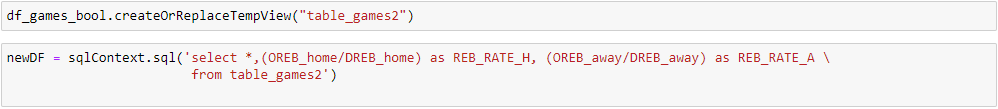
Our dataset is composed by 9234 rows and each row represents an NBA match with both the home team and the away team, and their statistics in the specific match starting from year 2014 to the present.

-Target variable creation:

once we had defined some potential features, we created a Boolean variable called “HOME\_WIN” that will represent our target of the prediction using the WL\_home variable.



-REB\_RATE:

The problem with our dataset is related to the nature of the variables that are available for us, because of their specificity related to a single event that do not provide real predictive value for our analysis. For this reason, we thought about another kind of feature that can provide some more information in making our prediction, the REB\_RATE. This new feature, calculate as the ratio between OREB and DREB, give us information about how much a team tend to be more offensive or defensive.

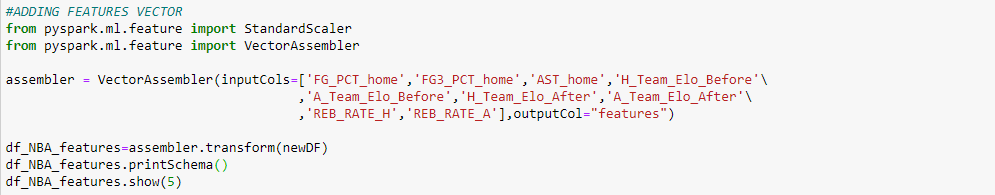
We could so include this variable in the features that are going to be given to the model.

-Model implementation:

The first step at this point was to create the vector of features and add it to our data-frame.

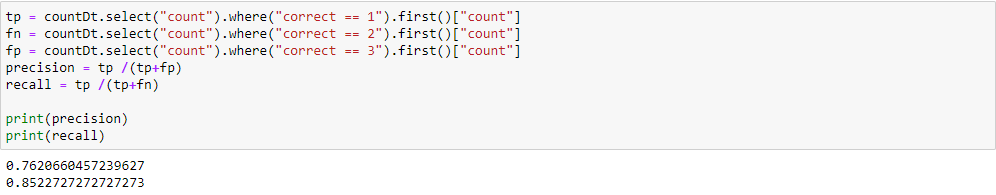
For choosing the better features we used the correlation matrix previously showed in order to take only the most correlated ones, and then of course we included the ELO ratings and the RATE\_REB.

Since that we had both the values for the home team and for the away team, we decide to take in consideration only the one related to the home team (a part for the REB\_RATE and the ELO ratings) because our real target is to predict if the Home team is going to win or not.



Once we scaled all the features inside the vector, in order to make them comparable, we divided our dataset randomly into training and test set, which respectively accounted 80 and 20 percent of the entire dataset.

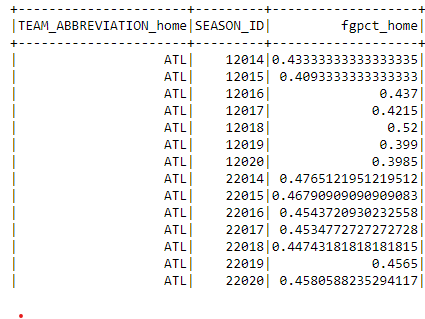
At this point we tried to apply both the logistic regression and the decision tree classifier to address our problem of classification and, after some trials with different features, we obtained the best performances, in terms of precision and recall, using the features below and the decision tree classifier.

We considered our outcome as a good one, by the way we would like to retrieve some variables that can have a possible predictive effect for our target, for this reason we addressed the following queries for that purpose.

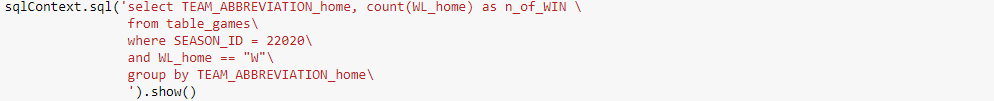
**Queries**

Our first query is aimed to retrieve the average accuracy for each home teams for each season:

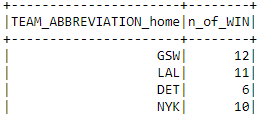
The output for one single team is the following:



The second query is aimed to retrieve the number of victories at home for each team during the current season, and it was coded directly using SQL syntax by creating a temporal view as we did before for creating the RATE\_REB variable:



Few rows of the output are the followings:



These two are only some examples we worked on, our entire code and additional queries we did can be viewed on Github thru the link we provided underneath.

|  |  |
| --- | --- |
| Group Members | Alessandro Pizzini, Gianpaolo Ceccarelli, Juan Riviera, Riccardo Sala, Leonhard Kossmann |
| Github Repository | <https://github.com/Pizzi95/Scalable-Data> |