A Work Project, presented as part of the requirements for the Award of a Master's degree in Management from the Nova School of Business and Economics.

Can Machine Learning algorithms predict Football players' Market Values?

A data-driven approach

Tiago Melo Dias, nº 26012

Work project carried out under the supervision of:

Carlos Daniel Santos, PhD

06-01-2020

Title: Can Machine Learning algorithms predict Football players' Market Values? A data-driven approach

Abstract: Football players transfer fees have been increasing in an unprecedented manner in the last decade. As so, it is crucial for football clubs to correctly assess the value of its players. To tackle this problem, this thesis proposes a data-driven solution. After assembling a dataset with data regarding football players' characteristics, on-field performance indicators and market values, Machine Learning algorithms were used to construct a prediction model. The final proposed model is a Random Forest regression, which registered a coefficient of determination (R^2) of 0.88 in the test set, displaying a promising outcome for future research.

Keywords: Football, Data analytics, Machine Learning, Management

This work used infrastructure and resources funded by Fundação para a Ciência e a Tecnologia (UID/ECO/00124/2013, UID/ECO/00124/2019 and Social Sciences DataLab, Project 22209), POR Lisboa (LISBOA-01-0145-FEDER-007722 and Social Sciences DataLab, Project 22209) and POR Norte (Social Sciences DataLab, Project 22209).

1. Introduction

Most association football, football hereafter, clubs' major source of income comes from players rights transactions, i.e. selling players to other clubs. In the past decade, players' transfer fees have been increasing in an unprecedented manner, due to larger availability of capital in the football realm, having the European market size reach € 28.4 billion in 2018 (Barnard et al., 2019). Nonetheless, players' market value estimates have not kept up with this upsurge, building a considerable discrepancy between what a club is willing to pay, and the experts' financial value of a player estimation. This situation creates an opportunity for data-driven systems to help football stakeholders, especially club management, to get a clearer assessment of the true monetary value of a football player.

The following thesis is organised as follows: literature review on using data analytics to predict players market values, methodology with detailed description of the machine learning techniques and algorithms employed and of the data used, discussion of the results and concluding notes. In brief, I will apply Machine Learning algorithms on football players' performance data and discrete information about themselves, the clubs and competitions they have represented and played. Then, through model comparison, regularisations and ensemble methods, I will propose a model that predicts a market value for football players.

2. Literature Review

The literature on using data analytics to estimate the financial value of a football player can almost be fully encompassed on the last two decades. Initial frameworks to determine the financial value of football players were mainly statistical models – i.e. regressions – based on variables like past transfer fees and player's performance indicators (Carmichael, Forrest and Simmons, 1999), but also on characteristics of the selling club and players wages (Lucifora and

Simmons, 2003). Extending on these, Tunaru, Clark and Viney (2005) proposed to treat players financial values as financial options, using stochastic calculus. This approach introduced a concept which was being neglected by previous models: players contracts are time bounded and during this period it is possible to exercise rights on the contract.

Subsequent research followed quantitative finance and econometric methodology, like Majewski and Majewska (2017), which instead of providing a value estimate, the model yielded a range of possible values, providing club management the ability of plan different strategies, based on distinct probable scenarios.

More recently, other approaches began to appear. Instead of just focusing on inside information, football fans' insights and experts' judgements on value estimation have also been analysed (Herm, Callsen-Bracker and Kreis, 2014), supporting the hypothesis that crowdsourcing judgements about a player's market value can influence the value of a hypothetical transfer fee. Despite being similar and correlated with the actual estimations of inside professionals, crowdsourcing estimation presents drawbacks like biased estimations and both lack of formality and validation (Müller, Simons and Weinmann, 2017). Müller, Simons and Weinmann (2017) proposed a data-driven approach towards market value estimation, leveraging player performance data with players characteristics and popularity, suggesting that football professionals were underestimating the data feeds provided by sports-data companies and in-house data collected by clubs themselves, using these just to improve on-field performance.

In what regards using Machine Learning, ML hereafter, techniques in football market values predictions, it is still in its embryonic stage, having only a handful of scholars applied these to the subject. Yiğit, Samak and Kaya (2020) used supervised ML techniques on videogames attributes of football players, recognising these as proxies of player quality. The

model was able to predict market values closer to transfer fees, when compared with crowdsourcing estimation. Kim, Bui and Jung (2019) produced a two-dimensional classification model, assessing player performance and transfers fees using player statistics data. On the other hand, sports-data companies like Opta Sports (Opta Sports, 2019) and InStat (InStat, n.d.) have proprietary software that may include ML techniques. In addition, KPMG offers a player valuation benchmark tool in a partnership with Opta Sports (KPMG, 2019).

3. Methodology

To perform this study, data regarding football players' basic information, performance, and market value was retrieve by implementing web scraping techniques, using Python programming language. After performing data curation - i.e. cleaning, filtering, wrangling and storage - descriptive statistics and data visualisation techniques were produced to assess data quality and appropriateness. The Scikit-Learn software library was used for data pre-processing and to assemble supervised ML algorithms to build market values prediction models. The algorithms used were Ordinary Least Squares (OLS) and Classification and Regression Tree (CART). On top of these algorithms, regularisations and ensemble learning methods were also implemented. Through model comparison, the model with better performance was selected as the most appropriate to produce football players' market values predictions.

Overall, it is suitable to follow such methodology because I have access to a large quantity of good quality data that allows me to make good features engineering, and the sample data, i.e. training-data, is a good representation of the population of professional football players in the most valuable football leagues.

3.1 Machine Learning algorithms and techniques

Like most ML problems, no algorithm is bound to out-perform any other algorithm. By ensuring data quality and appropriateness, simple models can produce very good results. I decided to test two different algorithms with the default Scikit-Learn parameters and hyperparameters, each one representing a different category of algorithms.

Ordinary Least Squares regression (OLS) – Linear regression model

Commonly known as linear regression, it tries to fit a line which minimises the residual sum of squares (RSS) of the input variables (the features) and the output variable (the label). The goal is to minimise the sum of the squared differences between the model predictions and actual label values. It seeks to identify linear relationship between the output variable with input variables.

Ridge and Lasso regressions – Regularisations

Also known as shrinking methods, while Lasso penalises the weights of less important variables by setting their coefficients to zero, i.e. L1 regularisation, Ridge regression also penalises the weights of less important variables, but the coefficients are always greater than zero, i.e. L2 regularisation (Géron, 2017). These can be seen as automatic feature selectors, especially the Lasso regularisation.

Classification and Regression Tree (CART) – Regression tree

Based on simple decision trees, "by learning decision rules inferred from the data features" it tries to predict the target variable value (Scikit-Learn, 2019). Respecting a top-down framework from a root node, the algorithm uses recursive partitioning, i.e. continuously splits the dataset, until no additional information gain results from splitting a node. This algorithm can identify non-linear relationships between features and label.

Random Forest – Bagging ensemble learning

The Random Forest ensemble method "builds multiple decision trees and merges them together to get a more accurate and stable prediction." (Géron, 2017). The final output is the average of outputs of the several decision trees, that have attributes and features randomly selected. By combining several independent learners, i.e. trees, and averaging their prediction, the algorithm produces trees with less variance, thus reducing the risk of overfitting the model to the training set.

3.2 Data description

All data was retrieved from transfermarkt.com (Transfermarkt, n.d.). Transfermarkt is a digital football platform that provides facts and figures about football and its partakers. Its database expands from simple in-game statistics to market values estimations. The provided market value estimates start at the 2004/2005 (or 2005) season, but just to specific leagues.

The data retrieved goes across 14 years, from 2006 to 2019, representing 13 football seasons. It is comprised by discrete information, per-season performance data and per-season market values estimation of football players. In total, data about 11 340 players from 22 leagues (14 European, 5 American and 3 Asian) across 13 seasons was scraped, resulting in a dataset with 40 173 records and 67 features. Leagues selection respected the following criteria: country-wise top division, having a total sum of players market value of at least € 250 million, as of September 2019. The leagues used are the top division from The United Kingdom, Spain, Germany, Italy, France, Portugal, Russia, The Netherlands, Brazil, Argentina, Belgium, Turkey, Austria, Ukraine, Greece, Switzerland, China, Japan, Saudi Arabia, Mexico, The United States of America and Colombia.

In what regards the dataset's features, discrete information about players includes their nationality, age, height in centimetres, on-field position, on-field main position, preferred foot, player's agent, outfitter (i.e. main sports sponsor), and also the club and national league they represented and played during the season. Some of these are categorical variables that suffered some transformation, which will be addressed in the following sections. Performance indicators are composed by club games, player played games, estimate of the points the club achieved to which the player contributed, goals scored, assists for goals made, own goals scored, times it got substitute on and off pitch, times it received a yellow, second yellow and red card, penalty goals and the total amount of minutes played. These 13 variables are recorded four times in the dataset, each time corresponding to different types of competitions, being national league, UEFA Champions League (UCL), other international competitions and other competitions. The remaining five features are the season in cause, two binary features stating whether the player played in the UCL and in other international competitions, and two market values features, being previous market value, i.e. at the beginning of the season, and market value, i.e. at the end of the season.

Whilst football players can be divided into four categories according with their playing area, being goalkeepers, defenders, midfielders and forwards, due to the characteristics of the collected performance data, only midfielders and forwards were analysed.

3.3 Priors, assumptions and data transformations

While building the dataset, some assumptions were made. If a player played in two different clubs or leagues during a season, it was assumed that he only played for the last club and league, having the in-game statistics of previous clubs been placed in other competitions. Player's agents and outfitters remain constant over time. The UEFA Champions League and

other international competitions have a different weight than other competitions, like national cups, so they their observations are not grouped with those competitions.

In order to use ML algorithms and to improve data quality and appropriateness, some mechanisms were employed. Categorical features observations were clustered into small groups, since most of these features had many different observations. The initial dataset had 22 different leagues, 717 clubs, 156 nationalities, 1778 player agents and 16 outfitters. Clubs were grouped based on the UEFA Club coefficients ranking for each season: the top five clubs were assigned with "UEFA Top 5", the top six to ten with "UEFA Top 10", the top 11 to 20 with "UEFA Top 20", while all other clubs were grouped into "Other". To nationalities, the clustering was based on the market value of the corresponding national teams, i.e. country squads, as of October 2019: the top 20 nationalities were left untouched, while the outstanding ones were categorised as "Other". Player agents were sorted by the total sum of market values of the players the agency represents, having the nomenclature of "Top 5 Agent", "Top 10 Agent" and "Top 20 Agent" for the 20 most valuable agencies, "Other" for the remaining agencies and "No agent" for null observations. Outfitters were rearranged as "Top Brand" for the two most representative brands, "Other" for the outstanding ones and "No outfitter" for null observations. Lastly, leagues, foot, on-field position, on-field main position, played UCL and played other international competitions did not suffer any kind of transformation.

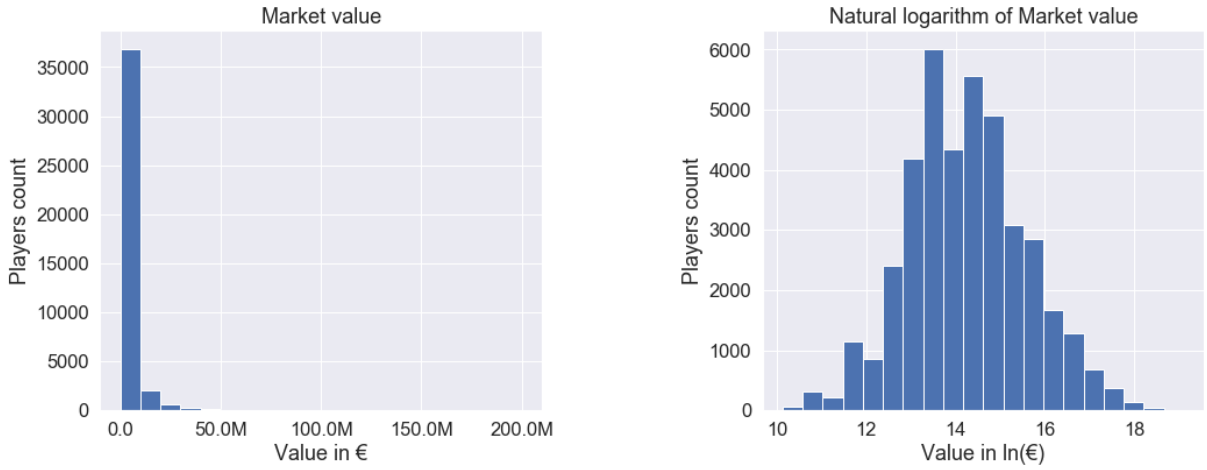
3.4 Data pre-processing and feature engineering

Before training any model, descriptive statistics and data visualisation techniques like correlation matrixes and distribution plots were produced to access data appropriateness. Analysing these provided important insights: some competition-based features (e.g. UCL features) were strongly correlated with each other, registering Pearson correlation coefficients

above 0.6. Regarding the label “Market value”, only “Previous market value” was highly correlated with it, with a correlation coefficient of 0.9. This does not come as a surprise, since it is expected that market values suffer from the anchoring cognitive bias of previous assessments (Lewicki, Barry and Saunders, 2010). Due to such high correlation, it was decided to assess two different scenarios: how ML algorithms would perform on predicting market value estimates knowing and not knowing the immediate previous judgment.

The “Market value” label also displayed high skewness to the right, with a positive skew of 7.17. To make the identification of patterns easier and more readable, natural logarithmic transformation was applied to the label “Market value”. Graphic 1 shows the effect of the applied transformation.

Graphic 1 – Effect of natural logarithm on Market value



Note: Market value distribution before and after logarithmic transformation, where is possible to see that most players have market values under € 10 million. Each bar on the left chart corresponds to a bin of € 10 million.

Additionally, manual feature engineering and selection was performed on the same competition-based features. In total, 16 new features were created, four for each type of competition (i.e. national league, UCL, other international competition and other competitions),

and 36 features were removed, nine for each type of competition. The new features were formalised as follows: “Minutes per game” by dividing the amount of minutes by the amount of games played, “Points per game” by dividing the amount of points by the amount of games played, “Disciplinary points” by combining the amount of yellow and red cards, but attributing more weight to red ones, and “Number of substitutions” by summing the times a player got substitute on and off. Equations 1, 2, 3 and 4 display the generic formalisation of “Minutes per game”, “Points per game”, “Disciplinary points” and “Number of substitutions” respectively.

Equation 1 – Minutes per game formulation

$$Minutes\ per\ game = \frac{Minutes\ played}{Games\ played}$$

Equation 2 – Points per game formulation

$$Points\ per\ game = \frac{Points}{Games\ played}$$

Equation 3 – Disciplinary points formulation

$$Disciplinary\ points = Yellow\ cards + (Second\ Yellow\ cards + Red\ cards) * 2$$

Equation 4 – Number of substitutions formulation

$$Number\ of\ substitutions = Substitutions\ on + Substitutions\ off$$

Consequently, most features used to generate the new ones were removed from the dataset. With the addition of “Own goals” and “Club games”, the features removed were “Minutes played”, “Yellow cards”, “Second yellow cards”, “Red cards”, “Points”, “Substitutes on” and “Substitutes off”, for each type of competition. “Own goals” were removed because these are misfortunate events that do not affect the assessment of a player quality, and “Club games” features were highly correlated with most features from the same competition, not providing sufficient new information. Once feature engineering and selection was finalised, the dataset was randomly split into two: a training set and a test set. The training set accounted for 80% of the total dataset, while the test set for 20%.

Further data transformations had to be executed before applying ML algorithms. These transformations include encoding categorical features. To encode categorical variables, the one-hot-encoding method was chosen. This method replaces the categorical feature with new binary features, one for each category (Scikit-Learn, 2019). As example, the “Club” feature had four possible categories: “UEFA Top 5”, “UEFA Top 10”, “UEFA Top 20” and “Other”. By applying the one-hot-encoding scheme, four new features are created to replace the “Club” feature: “Club_UEFA_Top_5”, “Club_UEFA_Top_10”, “Club_UEFA_Top_20” and “Club_Other”. For every instance in the dataset, if the “Club” value was “UEFA Top 5”, to “Club_UEFA_Top_5” is attributed the value “1”, and to “Club_UEFA_Top_10”, “Club_UEFA_Top_20” and “Club_Other” the value “0”. The same process is applied for the remaining values of the “Club” feature: “UEFA Top 10”, “UEFA Top 20” and “Other” attributed the value of “1” to ““Club_UEFA_Top_10”, “Club_UEFA_Top_20” and “Club_Other”, respectively, and “0” for the outstanding three features. By applying the one-hot-encoding, the nine initial categorical features were replaced by 56 binary features. As a result, a total of 111 features composed the transformed dataset.

4. Results and Discussion

The transformed training dataset was fitted in the two algorithms. To assess which one is more suitable to the dataset two metrics were selected: Root Mean Square Error (RMSE) and Coefficient of determination (R^2). Equation 5 and Equation 6 show the mathematical expressions of RMSE and R^2, respectively.

Equation 5 – Root Mean Square Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(y_j - \hat{y}_j\right)^2}$$

$$where, \quad n = sample\ size, \quad y_j = observed\ value, \quad \hat{y}_j = predicted\ value$$

Equation 6 – Coefficient of determination

$$R^2 = 1 - \frac{\sum_{j=1}^{n}\left(y_j - \hat{y}_j\right)^2}{\sum_{j=1}^{n}\left(y_j - \bar{y}\right)^2}$$

$$where, \quad n = sample\ size, \quad y_j = observed\ value, \quad \hat{y}_j = predicted\ value,$$

$$\bar{y} = mean\ of\ observed\ values$$

While RMSE measures the average difference between the estimate and the real value, in the same unit scale (i.e. lower is better), the R^2 measures the proportion of variance of the target variable explained by the predictor variables (i.e. higher is better). Table 1 summarises the findings of applying the two algorithms, using all possible features. The only restriction made was on the CART algorithm, imposing early stopping at a depth of 10 nodes.

Table 1 – RMSE and R^2 scores for the two algorithms, on the first training

Scenario	Algorithm	RMSE	R^2
Using "Previous market value"	Ordinary Least Squares	0.70	0.73
	Classification and Regression Tree	0.46	0.88
Not using "Previous market value"	Ordinary Least Squares	0.75	0.69
	Classification and Regression Tree	0.80	0.65

Two main conclusions can be drawn. First, as expected, "Previous market value" has a significant impact on the algorithms' performance. Secondly, the CART decision tree seems to be underfitting if "Previous market value" is not used. This is a result of early stopping. If no restrictions were made, the CART would most likely overfit the training set. The algorithm would have designed a very complex tree that could predict any instance in the training set, making it hard to generalised for new data points.

To assess overfitting in both the OLS and CART algorithms, some mechanisms were adopted. To the OLS, two types of regularisations, i.e. shrinking methods, were used and computed: Lasso and Ridge regularised regression versions. In what regards the CART, the Random Forest bagging ensemble method was select. Table 2 indicates the results of the regularisations and Random Forest.

Table 2 – RMSE and R^2 scores for regularised regressions and Random Forest, on the first training

Scenario	Algorithm	RMSE	R^2
Using "Previous market value"	Lasso Regression	0.93	0.53
	Ridge Regression	0.69	0.73
	Random Forest	0.42	0.91
Not using "Previous market value"	Lasso Regression	1.07	0.37
	Ridge Regression	0.74	0.69
	Random Forest	0.74	0.70

Based on these metrics, it is possible to see that the regularised linear regression models have different performances. Ridge regularisation clearly outperforms the Lasso one, meaning that less important variables play a significant role. By the Ridge result, it seems that the OLS model is not overfitting the training set. On the other hand, the Random Forest was able to outperform the CART algorithm, with the same level of depth, and with 200 estimators, i.e. number of trees.

In order to select the most appropriate model, a 10-fold cross validation evaluation was performed. This method randomly splits the training set into 10 distinct subsets, called folds. Then it trains and evaluates the models 10 times, picking a different fold for every evaluation and training on the other nine folds. The result is an array containing the 10 evaluation scores. The measure selected to assess how good the model generalises was the RMSE. Table 3 displays the results.

Table 3 – Results from the 10-fold cross validation

Scenario	Algorithm	Mean RMSE	RMSE Standard deviation
Using "Previous market value"	Ordinary Least Squares	0.70	0.02
	Classification and Regression Tree	0.56	0.01
	Random Forest	0.49	0.01
Not using "Previous market value"	Ordinary Least Squares	0.75	0.01
	Classification and Regression Tree	0.88	0.01
	Random Forest	0.79	0.01

If "Previous market value" is known, the Random Forest model clearly outperformed, on average, the other two models across the 10 folds. If "Previous market value" is unknown, the OLS outperformed, on average, the remaining models, but closely followed by the Random Forest. Since that there is such a difference in performance in the first scenario, and the standard deviations on both settings are similar across the models, the Random Forest was the model selected.

The following step was to fine tune the Random Forest hyperparameters. To perform this task, a 10-fold cross-validation grid search was executed. The selection criteria was again the RMSE, having the grid two hyperparameters to optimise: the maximum number of features the algorithm can consider to find the best split at each node and the number of trees in the forest. The proposed values were [10, 50, "auto"] for the maximum features and [100, 200, 500] for the number of trees in the forest, where "auto" means all features. All other hyperparameters

values remained as the default Scikit-Learn values. Chart 1 shows all the hyperparameters of the final proposed tuned model, which result from the 10-fold cross-validation grid search.

Chart 1 - Random Forest model hyperparameters after tuning

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, max_features='auto', max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=4, oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Note: This is a detailed description of the hyperparameters used on the final model. This piece of code runs on Python, through the Scikit-Learn library, if instantiated, creating a Random Forest regression model.

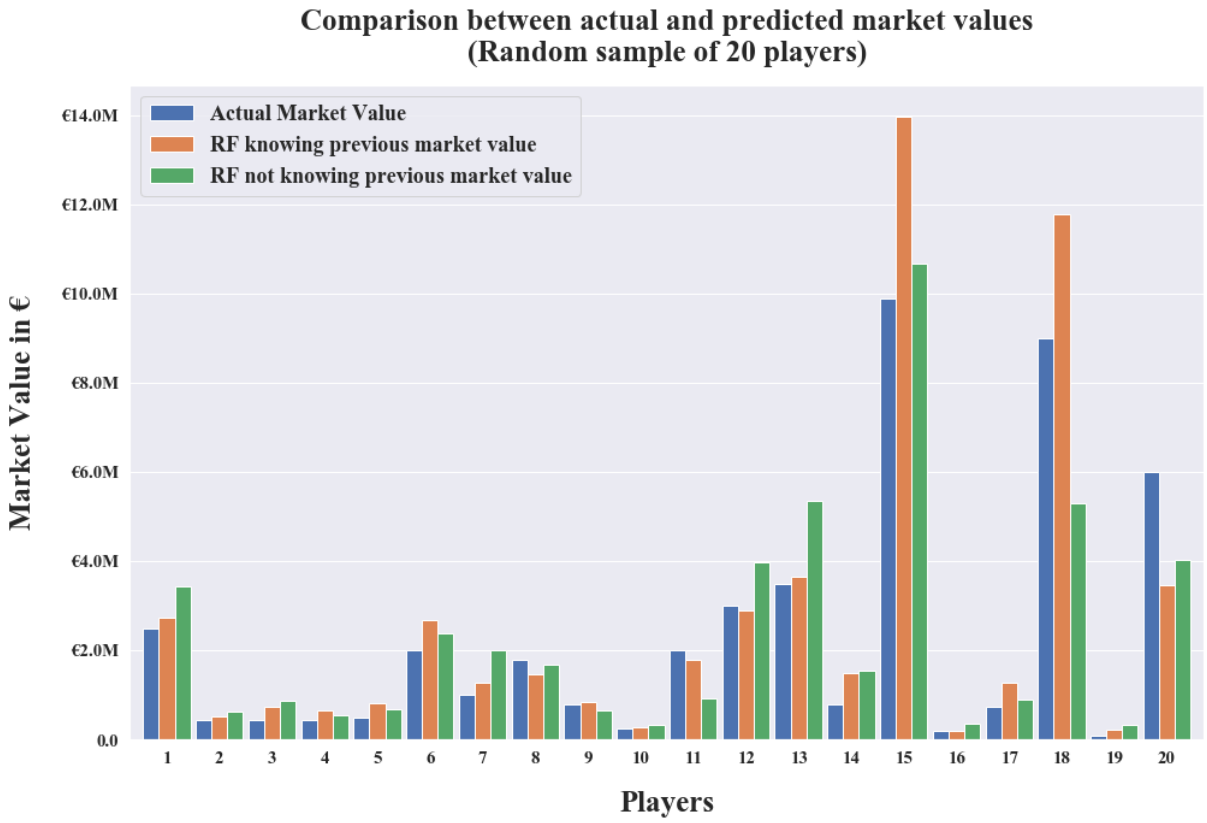
The final stage was to predict market values on the test set. Results are displayed in Table 4.

Table 4 – Model predictions on the test set metrics

Scenario	Model	RMSE	RMSE 95% confidence interval	R^2
Using "Previous market value"	Random Forest	0.46	[0.45 : 0.47]	0.88
Not using "Previous market value"	Random Forest	0.68	[0.66 : 0.69]	0.74

In both scenarios the Random Forest model registered a good performance on the test set. A sample comparison between the predicted and actual market values can be seen in Graphic 2.

Graphic 2 – Comparison between actual and predicted market values



Note: Each number represents a specific player. It is possible to see that in both scenarios the model yields proper predictions. To get these predictions in Euros, the natural logarithm transformation had to be reverted, by applying its inverse function to the Random Forest outputs.

Although the predictions consider all features, depending on the situation, the Random Forest model attributes different importance to different features to make node decision. Table 5 and 6 show the top five features and their relative importance to the model, if "Previous market value" is known or unknown, respectively.

Table 5 – Random Forest feature importance, if previous market value is known

Feature	Importance (%)
Previous market value	78.38
Age	3.05
League Games	2.84
League Minutes per game	1.37
League Points per game	1.20

Note: These are the first features the model looks for to make node local decision.

Table 6 – Random Forest feature importance, if previous market value is unknown

Feature	Importance (%)
UCL Minutes per game	17.56
League Games	12.63
Age	4.97
League Points per game	4.45
League Goals	4.32

Note: These are the first features the model looks for to make node local decision.

4.1 Limitations

There are two main drawbacks in the dataset, being the frequency of instances and the small range of performance features. Instances are annual, meaning that for one player the maximum number of observations per season is just one. This situation has to do with the frequency to which market values are updated, and also how these updates are made – not all

players have their market values updated at the same point in time. Regarding performance data, it is strict to offensive (goals and assists), disciplinary display (yellow and red cards) and play time (minutes and substitutions). This hindered the possibility of analysing individual performance indicators with great detail. Indicators related with other on-field actions like passes, tackles and dribbles are expected to increase the robustness of the model.

5. Conclusions

This thesis is an end-to-end project that started from identifying possible data feeds sources, passed through assembling a dataset from scratch and produce ML supervised models capable of predicting football players market values. After completing the whole process, the fundamental question still is: can machine learning algorithms predict football players' market values? The answer is not obvious. The proposed model is able to predict market value estimates with a coefficient of determination of 0.88 in the test set. Although the model decreased its performance on the test set when compared with the results on the training set, if previous market value assessments were unknown the model was able par its test and training performances.

In fact, the key point in this study was to try to explain what can influence the value attributed to a football player. The first main conclusion is that what influences the most are previous assessments. The market values estimates used in this thesis were crowdsourced-based. Once a football player is recognised as valuable, the anchoring cognitive bias towards that value is activated.

Secondly, the model gives more importance to success-based features rather than players' characteristics. Only to age was given a close relative importance as to play time and winning games, i.e. points per game. This fact goes into accordance with the colloquial

knowledge that many clubs value their players not just on past performance, but also on potential future performance, appraising young players as more prone to display such potential ability. Yet, it was expected that age would play a larger role, i.e. more relative importance, in assessing the value of a player.

The last main conclusion is that estimating a player's market value is not a simple task. Other instances like popularity, influence and professionalism were not tackled. For example, superstar players cannot just be valued by their on-field performance. These players not only exert influence over the fans, but also on the business side of football, e.g. sponsorships. Nonetheless, most football players market values do not reach the € 10 million mark. This was observed when the skewness of the "Market value" feature was analysed. Although, the high transfer fees were the ones that inspired this study, it is crucial for clubs' management, both on the buying and selling side, to effectively establish a threshold, and not rely just on crowdsource assessments.

In what regards the model, two different approaches could be tried to improve it: gather more data and better feature selection. Nevertheless, these are hard to implement. Getting access to better data its costlier. As previously stated, clubs and sports-data providers collect and have access to more granular and diverse data. With such data better feature engineering can be achieved, but the level of noisy data also increases. This would imply applying other mechanisms, like dimensionality reduction of the dataset.

Lastly, the applied algorithms have a small level of complexity. Future research should focus on using more complex algorithms, like Artificial neural networks (ANN), which could achieve better results. Yet, these deep learning algorithms create black-box models, in which there is little knowledge on how these models make predictions.

6. References

Barnard, Michael, Sam Boor, Christopher Winn, Chris Woodm, and Izzy Wray. 2019. “Annual Review of Football Finance 2019”. Deloitte LLP.

Carmichael, Fiona, David Forrest, and Robert Simons. 1999. “The Labour Market in Association Football: Who Gets Transferred and for How Much?”. *Bulletin of Economic Research*, 51(2): 125–150.

Géron, Aurélien. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol: O’Reilly Media , Inc..

Herm, Steffen, Hans-Markus Callsen-Bracker, and Henning Kreis. 2014. “When the crowd evaluates soccer players’ market values: Accuracy and evaluation attributes of an online community”. *Sport Management Review*, 17(4): 484–492.

InStat Sports. n.d. “Football.” Accessed November 8, 2019. https://instatsport.com/football.

Kim, Yunhu, Khac-Hoai Nam Bui, and Jason J. Jung. 2019. “Data-driven exploratory approach on player valuation in football transfer market”. *Concurrency and Computation: Practice and Experience*, e5353.

KPMG. 2019. “Player Valuation.” Accessed November 8, 2019. https://www.footballbenchmark.com/methodology/player_valuation.

Lewicki, R.J., Bruce Barry, and David Saunders. 2010. *Negotiation*. New York: McGraw-Hill.

Lucifora, Claudio, and Rob Simmons. 2003. “Superstar Effects in Sport: Evidence From Italian Soccer”. *Journal of Sports Economics*, 4(1): 35–55.

Majewski, Sebastian, and Agnieszka Majewska. 2017. “Using Monte Carlo Methods for the Valuation of Intangible Assets in Sports Economics”. *Folia Oeconomica Stetinensia*, 17(2): 71–82.

Müller, Oliver, Alexander Simons, and Markus Weinmannb. 2017. “Beyond crowd judgments: Data-driven estimation of market value in association football”. *European Journal of Operational Research*, 263(2): 611–624.

Opta Sports. 2019. “Data feeds.” Accessed November 8, 2019. https://www.optasports.com/services/data-feeds/.

Scikit-Learn. 2019. “User guide.” Accessed December 18, 2019. https://scikit-learn.org/stable/user_guide.html.

Transfermarkt. n.d. “Football Transfers, Rumours, Market Values, News and Statistics.” Accessed December 20, 2019. https://www.transfermarkt.com/.

Tunaru, Radu, Ephraim Clark, and Howard Viney. 2005. “An option pricing framework for valuation of football players”. *Review of Financial Economics*, 14: 281–295.

Yiğit, Ahmet, Barış Samak, and Tolga Kaya. 2020. “Football Player Value Assessment Using Machine Learning Techniques”. In *Intelligent and Fuzzy Techniques in Big Data Analytics and Decision Making*, edited by Cengiz Kahraman, Selçuk Cebi, Sezi Cevik Onar, Basar Oztaysi, A. Cagri Tolga, and Irem Ucal Sari, 289 – 297. Istanbul: Springer International Publishing.

7. Appendices

Appendix 1 – Python code used for applying Machine Learning

Standard imports

In [1]:
```
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 150
pd.options.display.max_rows = 150
pd.options.display.float_format = '{:.3f}'.format
from datetime import datetime
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator, MultipleLocator, FuncFormatter
import seaborn as sns
sns.set()
import scipy
import sklearn
from scipy import stats
np.random.seed(42)
```

Import dataset

In [2]:
```
data = pd.read_csv('Final dataset.csv', squeeze=True, index_col=0)
```

Dataset information

In [4]:
```
print(f'Number of records = {data.shape[0]}\nNumber of variables = {data.shape[1]}')
```

```
Number of records = 40173
Number of variables = 67
```

In [5]:
```
print(f'List of variables and their data type:\n\n{data.dtypes}')
```

```
List of variables and their data type:

Season                                  int64
League                                 object
Club                                   object
League Club games                       int64
League Games                            int64
League Points                         float64
League Goals                            int64
League Assists                          int64
League Own goals                        int64
League Subs on                          int64
League Subs off                         int64
League Yellow cards                     int64
League Second yellow cards              int64
League Red cards                        int64
League Penalty goals                    int64
League Minutes played                   int64
Height (in cm)                        float64
Nationality                            object
Position                               object
Main position                          object
Foot                                   object
Player agent                           object
Outfitter                              object
Age                                   float64
Market value                          float64
Previous market value                 float64
Other competitions Club games         float64
Other competitions Games              float64
Other competitions Points             float64
Other competitions Goals              float64
Other competitions Assists            float64
Other competitions Own goals          float64
Other competitions Subs on            float64
Other competitions Subs off           float64
Other competitions Yellow cards       float64
Other competitions Second yellow cards float64
Other competitions Red cards          float64
Other competitions Penalty goals      float64
Other competitions Minutes played     float64
UCL Club games                        float64
UCL Games                             float64
UCL Points                            float64
UCL Goals                             float64
UCL Assists                           float64
UCL Own goals                         float64
UCL Subs on                           float64
UCL Subs off                          float64
UCL Yellow cards                      float64
UCL Second yellow cards               float64
UCL Red cards                         float64
UCL Penalty goals                     float64
UCL Minutes played                    float64
Played UCL                             object
Other int. comp. Club games           float64
Other int. comp. Games                float64
Other int. comp. Points               float64
Other int. comp. Goals                float64
Other int. comp. Assists              float64
Other int. comp. Own goals            float64
Other int. comp. Subs on              float64
Other int. comp. Subs off             float64
Other int. comp. Yellow cards         float64
Other int. comp. Second yellow cards  float64
Other int. comp. Red cards            float64
Other int. comp. Penalty goals        float64
Other int. comp. Minutes played       float64
Played Other int. comp.                object
dtype: object
```

Categorical variables

Nationality

In [6]:
```
print(f'''Number of categories: {len(np.sort(data['Nationality'].unique()))}''')
data['Nationality'].value_counts()
```

```
Number of categories: 21
```

Out[6]:
```
Other          15448
Brazil          4140
Argentina       2194
Spain           1916
France          1860
Italy           1715
Turkey          1630
Netherlands     1557
Portugal        1304
Germany         1208
Belgium         1146
England         1065
Austria         1065
Colombia        1057
Switzerland      790
Serbia           507
Uruguay          460
Senegal          380
Croatia          370
Denmark          202
Poland           159
Name: Nationality, dtype: int64
```

Position

In [7]:
```
print(f'''Number of categories: {len(np.sort(data['Position'].unique()))}''')

data['Position'].value_counts()
```

```
Number of categories: 2
```

Out[7]:
```
Midfielder    21846
Forward       18327
Name: Position, dtype: int64
```

Main position

In [8]:
```
print(f'''Number of categories: {len(np.sort(data['Main position'].unique()))}''')

data['Main position'].value_counts()
```

```
Number of categories: 11
```

Out[8]:
```
Centre-Forward        10212
Defensive Midfield     7226
Central Midfield       7102
Attacking Midfield     4765
Right Winger           3693
Left Winger            3400
Left Midfield          1374
Right Midfield         1318
Second Striker          958
Forward                  64
Midfielder               61
Name: Main position, dtype: int64
```

Foot

In [9]:
```
print(f'''Number of categories: {len(np.sort(data['Foot'].unique()))}''')

data['Foot'].value_counts()
```

```
Number of categories: 3
```

Out[9]:
```
Right    28840
Left      7931
Both      3402
Name: Foot, dtype: int64
```

Player Agent

In [10]:
```
print(f'''Number of categories: {len(np.sort(data['Player agent'].unique()))}''')

data['Player agent'].value_counts()
```

```
Number of categories: 5
```

Out[10]:
```
Other           22497
No agent        14956
Top 20 Agent     1006
Top 5 Agent       974
Top 10 Agent      740
Name: Player agent, dtype: int64
```

Outfitter

In [11]:
```
print(f'''Number of categories: {len(np.sort(data['Outfitter'].unique()))}''')

data['Outfitter'].value_counts()
```

```
Number of categories: 3
```

Out[11]:
```
No outfitter    33806
Top brand        5379
Other             988
Name: Outfitter, dtype: int64
```

League

In [12]:
```
print(f'''Number of categories: {len(np.sort(data['League'].unique()))}''')

data['League'].value_counts()
```

```
Number of categories: 22
```

Out[12]:
```
Serie A                  3116
Premier League           3051
LaLiga                   2814
Ligue 1                  2797
Süper Lig                2517
Bundesliga               2504
Série A                  2422
Eredivisie               2262
Jupiler Pro League       2122
Liga NOS                 2078
Premier Liga             1948
MLS                      1797
Super League 1           1749
Liga MX Clausura         1698
Ukrainian Premier Liga   1521
Austrian Bundesliga      1285
Super League             1275
Chinese Super League      943
Professional League       789
Liga Águila II            574
Superliga                 526
J1 League                 385
Name: League, dtype: int64
```

Club

In [13]:
```
print(f'''Number of categories: {len(np.sort(data['Club'].unique()))}''')

data['Club'].value_counts()
```

```
Number of categories: 4
```

Out[13]:
```
Other          37005
UEFA Top 20     1621
UEFA Top 10      775
UEFA Top 5       772
Name: Club, dtype: int64
```

Played UCL

In [14]:
```
print(f'''Number of categories: {len(np.sort(data['Played UCL'].unique()))}''')

data['Played UCL'].value_counts()
```

```
Number of categories: 2
```

Out[14]:
```
No     35958
Yes     4215
Name: Played UCL, dtype: int64
```

Played Other int. comp.

In [15]:
```
print(f'''Number of categories: {len(np.sort(data['Played Other int. comp.'].unique()))}''')

data['Played Other int. comp.'].value_counts()
```

```
Number of categories: 2
```

Out[15]:
```
No     30622
Yes     9551
Name: Played Other int. comp., dtype: int64
```

Summary of the numerical attributes

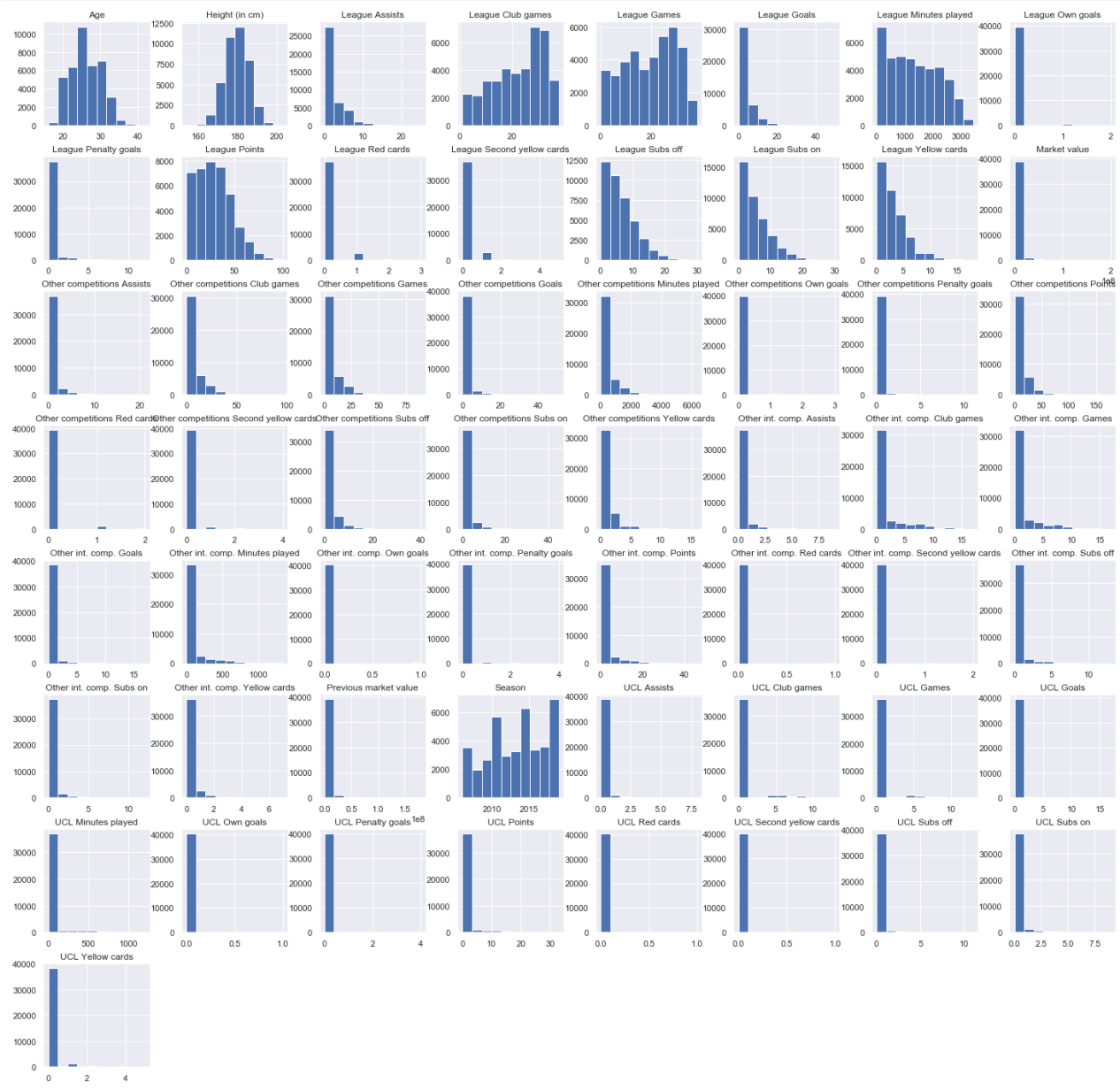
In [16]:
```
data.describe().transpose()
```

Out[16]:

	count	mean	std	min	25%	50%	75%	max
Season	40173.000	2013.270	3.816	2006.000	2010.000	2014.000	2017.000	2019.000
League Club games	40173.000	22.441	9.871	1.000	15.000	25.000	31.000	38.000
League Games	40173.000	19.383	10.160	0.000	11.000	20.000	28.000	38.000
League Points	40173.000	28.177	18.057	0.000	14.000	27.020	40.000	99.940
League Goals	40173.000	3.012	4.082	0.000	0.000	2.000	4.000	50.000
League Assists	40173.000	2.132	2.674	0.000	0.000	1.000	3.000	25.000
League Own goals	40173.000	0.016	0.128	0.000	0.000	0.000	0.000	2.000
League Subs on	40173.000	4.698	4.366	0.000	1.000	4.000	7.000	30.000
League Subs off	40173.000	5.555	4.571	0.000	2.000	5.000	8.000	30.000
League Yellow cards	40173.000	2.829	2.657	0.000	1.000	2.000	4.000	18.000
League Second yellow cards	40173.000	0.080	0.291	0.000	0.000	0.000	0.000	5.000
League Red cards	40173.000	0.070	0.269	0.000	0.000	0.000	0.000	3.000
League Penalty goals	40173.000	0.268	0.861	0.000	0.000	0.000	0.000	12.000
League Minutes played	40173.000	1310.431	878.533	0.000	549.000	1242.000	2038.000	3420.000
Height (in cm)	40173.000	179.534	6.179	154.000	175.000	180.000	184.000	203.000
Age	40173.000	25.989	3.993	16.000	23.000	26.000	29.000	42.000
Market value	40173.000	3928344.908	8297998.564	25000.000	600000.000	1500000.000	3750000.000	200000000.000
Previous market value	40173.000	3457459.737	7182414.508	25000.000	500000.000	1200000.000	3500000.000	180000000.000
Other competitions Club games	40173.000	6.953	7.822	0.000	2.000	4.000	9.000	96.000
Other competitions Games	40173.000	6.084	7.084	0.000	1.000	4.000	8.000	89.000
Other competitions Points	40173.000	9.902	11.886	0.000	1.000	6.000	14.000	177.810
Other competitions Goals	40173.000	1.195	2.423	0.000	0.000	0.000	1.000	51.000
Other competitions Assists	40173.000	0.673	1.378	0.000	0.000	0.000	1.000	21.000
Other competitions Own goals	40173.000	0.005	0.070	0.000	0.000	0.000	0.000	3.000
Other competitions Subs on	40173.000	1.454	2.376	0.000	0.000	1.000	2.000	45.000
Other competitions Subs off	40173.000	1.750	2.634	0.000	0.000	1.000	2.000	40.000
Other competitions Yellow cards	40173.000	0.836	1.479	0.000	0.000	0.000	1.000	16.000
Other competitions Second yellow cards	40173.000	0.026	0.169	0.000	0.000	0.000	0.000	4.000
Other competitions Red cards	40173.000	0.027	0.167	0.000	0.000	0.000	0.000	2.000
Other competitions Penalty goals	40173.000	0.106	0.474	0.000	0.000	0.000	0.000	11.000
Other competitions Minutes played	40173.000	420.447	521.482	0.000	90.000	244.000	530.000	6336.000
UCL Club games	40173.000	0.663	2.114	0.000	0.000	0.000	0.000	13.000
UCL Games	40173.000	0.568	1.879	0.000	0.000	0.000	0.000	13.000
UCL Points	40173.000	0.850	3.249	0.000	0.000	0.000	0.000	33.020
UCL Goals	40173.000	0.092	0.568	0.000	0.000	0.000	0.000	17.000
UCL Assists	40173.000	0.068	0.405	0.000	0.000	0.000	0.000	8.000
UCL Own goals	40173.000	0.000	0.019	0.000	0.000	0.000	0.000	1.000
UCL Subs on	40173.000	0.150	0.654	0.000	0.000	0.000	0.000	9.000
UCL Subs off	40173.000	0.161	0.694	0.000	0.000	0.000	0.000	11.000
UCL Yellow cards	40173.000	0.076	0.377	0.000	0.000	0.000	0.000	5.000
UCL Second yellow cards	40173.000	0.002	0.045	0.000	0.000	0.000	0.000	1.000
UCL Red cards	40173.000	0.001	0.031	0.000	0.000	0.000	0.000	1.000
UCL Penalty goals	40173.000	0.007	0.108	0.000	0.000	0.000	0.000	4.000
UCL Minutes played	40173.000	37.740	136.472	0.000	0.000	0.000	0.000	1200.000
Other int. comp. Club games	40173.000	1.277	2.762	0.000	0.000	0.000	0.000	17.000
Other int. comp. Games	40173.000	1.120	2.491	0.000	0.000	0.000	0.000	17.000
Other int. comp. Points	40173.000	1.754	4.462	0.000	0.000	0.000	0.000	46.070
Other int. comp. Goals	40173.000	0.177	0.723	0.000	0.000	0.000	0.000	17.000
Other int. comp. Assists	40173.000	0.123	0.517	0.000	0.000	0.000	0.000	9.000
Other int. comp. Own goals	40173.000	0.001	0.028	0.000	0.000	0.000	0.000	1.000
Other int. comp. Subs on	40173.000	0.286	0.868	0.000	0.000	0.000	0.000	12.000
Other int. comp. Subs off	40173.000	0.321	0.935	0.000	0.000	0.000	0.000	13.000
Other int. comp. Yellow cards	40173.000	0.157	0.531	0.000	0.000	0.000	0.000	7.000
Other int. comp. Second yellow cards	40173.000	0.005	0.070	0.000	0.000	0.000	0.000	2.000

	count	mean	std	min	25%	50%	75%	max
Other int. comp. Red cards	40173.000	0.004	0.063	0.000	0.000	0.000	0.000	1.000
Other int. comp. Penalty goals	40173.000	0.013	0.136	0.000	0.000	0.000	0.000	4.000
Other int. comp. Minutes played	40173.000	75.319	181.241	0.000	0.000	0.000	0.000	1336.000

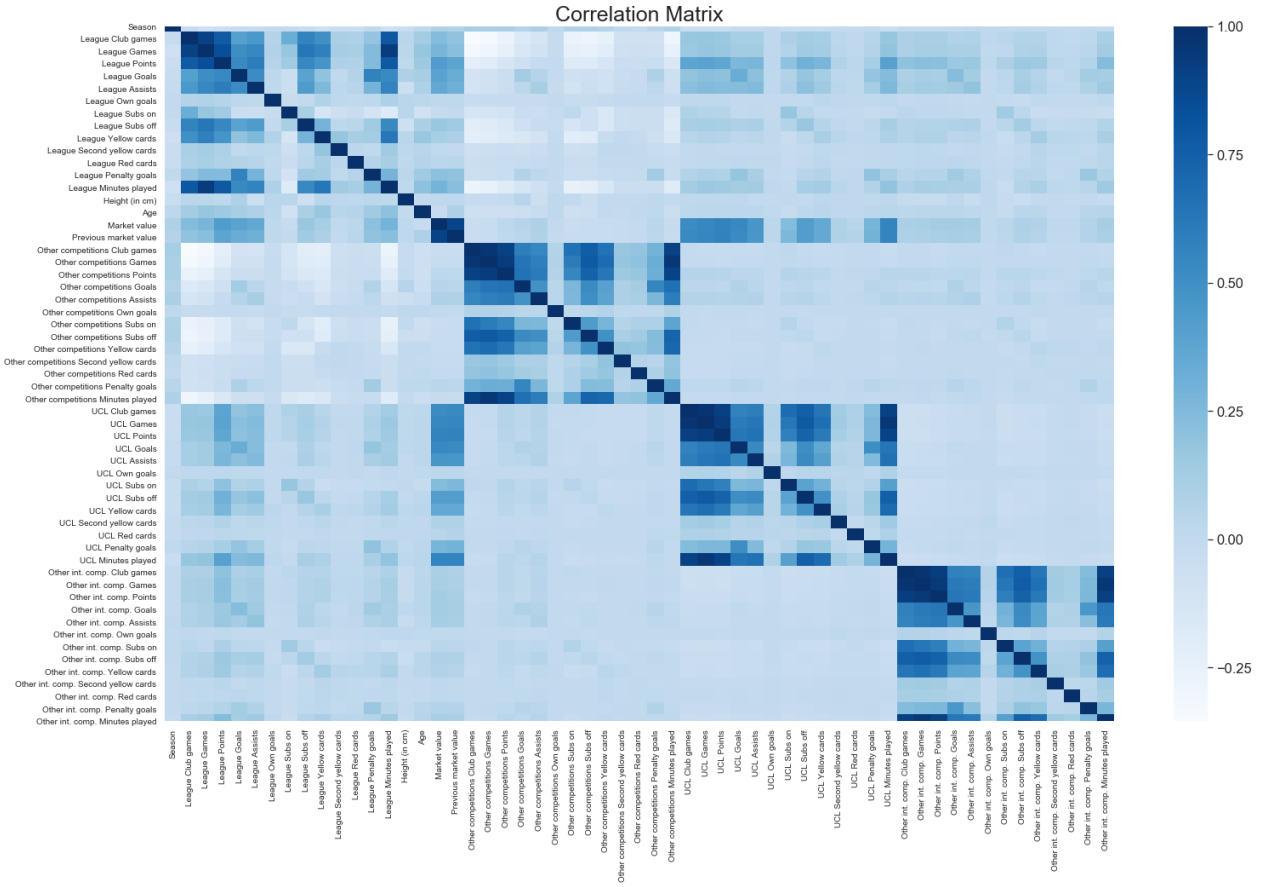
In [17]:
```
distributions = data.hist(figsize=(25,25))

plt.show()
```



In [18]:
```
correlations = data.corr()
```

In [19]:
```
sns.set(rc={'figure.figsize':(25,15)})
sns.set(font_scale=1.5)

mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

ax = sns.heatmap(correlations, cmap='Blues', yticklabels=correlations.index,
                 xticklabels=correlations.columns)


ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 10)
ax.set_xticklabels(ax.get_xticklabels(), rotation = 90, fontsize = 10)


ax.set_title('Correlation Matrix', fontsize = 25)
```

Out[19]:
```
Text(0.5, 1, 'Correlation Matrix')
```



In [20]:
```
correlations['Market value'].sort_values(ascending=False)
```

Out[20]:
```
Market value                          1.000
Previous market value                 0.901
UCL Points                            0.567
UCL Minutes played                    0.566
UCL Games                             0.548
UCL Goals                             0.541
UCL Club games                        0.519
UCL Assists                           0.469
League Points                         0.433
UCL Subs off                          0.425
League Goals                          0.394
League Assists                        0.360
UCL Yellow cards                      0.354
League Minutes played                 0.286
UCL Penalty goals                     0.281
League Games                          0.276
League Club games                     0.242
UCL Subs on                           0.241
League Penalty goals                  0.212
League Subs off                       0.173
Other int. comp. Goals                0.131
Other int. comp. Points               0.127
Other int. comp. Assists              0.123
League Yellow cards                   0.119
Other int. comp. Minutes played       0.116
Other int. comp. Games                0.107
Other int. comp. Club games           0.097
Other int. comp. Subs off             0.086
Season                                0.084
Other competitions Assists            0.082
UCL Second yellow cards               0.074
Other int. comp. Yellow cards         0.071
Other int. comp. Penalty goals        0.064
Other competitions Goals              0.052
UCL Red cards                         0.047
Other competitions Penalty goals      0.035
Other int. comp. Subs on              0.034
League Red cards                      0.028
UCL Own goals                         0.027
Other competitions Points             0.022
League Own goals                      0.020
Other int. comp. Red cards            0.012
Other int. comp. Second yellow cards  0.012
Height (in cm)                        0.011
League Second yellow cards            0.011
Other int. comp. Own goals            0.007
Other competitions Own goals         -0.005
Other competitions Red cards         -0.009
Other competitions Second yellow cards -0.012
Age                                  -0.014
Other competitions Minutes played    -0.017
League Subs on                       -0.021
Other competitions Yellow cards      -0.027
Other competitions Subs off          -0.028
Other competitions Games             -0.029
Other competitions Subs on           -0.039
Other competitions Club games        -0.040
Name: Market value, dtype: float64
```

In [21]:
```
correlations
```

Out[21]:

	Season	League Club games	League Games	League Points	League Goals	League Assists	League Own goals	League Subs on	League Subs off	League Yellow cards	League Second yellow cards	League Red cards	League Penalty goals	League Minutes played	Heig (cr
Season	1.000	-0.015	-0.074	-0.053	-0.025	-0.012	-0.003	-0.004	-0.024	-0.059	-0.038	-0.036	0.007	-0.072	-0.03
League Club games	-0.015	1.000	0.914	0.773	0.412	0.453	0.063	0.324	0.568	0.501	0.097	0.083	0.189	0.790	0.03
League Games	-0.074	0.914	1.000	0.844	0.512	0.548	0.072	0.180	0.631	0.586	0.123	0.107	0.248	0.932	0.02
League Points	-0.053	0.773	0.844	1.000	0.541	0.598	0.058	0.147	0.548	0.471	0.087	0.078	0.247	0.787	0.01
League Goals	-0.025	0.412	0.512	0.541	1.000	0.522	0.023	-0.046	0.393	0.200	0.010	0.063	0.572	0.531	0.08
League															

In [22]:
```
data.skew().sort_values(ascending=False)
```

Out[22]:
```
UCL Own goals                              51.724
Other int. comp. Own goals                 35.391
UCL Red cards                              32.049
UCL Second yellow cards                    22.067
UCL Penalty goals                          19.374
Other competitions Own goals               16.472
Other int. comp. Red cards                 15.751
Other int. comp. Second yellow cards       15.009
Other int. comp. Penalty goals             12.477
UCL Goals                                  10.351
UCL Assists                                 8.519
League Own goals                            8.193
Other competitions Second yellow cards      7.404
Market value                                7.172
Previous market value                       6.951
Other competitions Penalty goals            6.950
Other competitions Red cards                6.510
Other int. comp. Goals                      6.431
UCL Yellow cards                            6.106
Other int. comp. Assists                    5.986
UCL Subs off                                5.635
UCL Subs on                                 5.606
UCL Points                                  4.747
League Penalty goals                        4.504
Other competitions Goals                    4.328
Other int. comp. Yellow cards               4.286
UCL Minutes played                          4.191
Other int. comp. Subs on                    4.095
League Red cards                            3.943
Other int. comp. Subs off                   3.933
League Second yellow cards                  3.890
UCL Games                                   3.600
Other competitions Assists                  3.570
UCL Club games                              3.352
Other int. comp. Points                     3.211
Other competitions Subs on                  3.083
Other competitions Yellow cards             2.940
Other int. comp. Minutes played             2.932
Other competitions Subs off                 2.818
Other int. comp. Games                      2.509
League Goals                                2.490
Other int. comp. Club games                 2.325
Other competitions Minutes played           2.282
Other competitions Points                   2.160
Other competitions Games                    2.015
League Assists                              1.944
Other competitions Club games               1.885
League Subs on                              1.178
League Yellow cards                         1.129
League Subs off                             0.952
League Points                               0.471
Age                                         0.288
League Minutes played                       0.223
Height (in cm)                             -0.050
Season                                     -0.217
League Games                               -0.218
League Club games                          -0.457
dtype: float64
```

Feature engineering

Minutes per game

In [23]:
```
data['League Minutes per game'] = ((data['League Minutes played']) /
                                   (data['League Games']))

data['UCL Minutes per game'] = ((data['UCL Minutes played']) /
                                (data['UCL Games']))

data['Other int. comp. Minutes per game'] = ((data['Other int. comp. Minutes played']) /
                                             (data['Other int. comp. Games']))

data['Other competitions Minutes per game'] = ((data['Other competitions Minutes played']) /
                                               (data['Other competitions Games']))
```

Points per game played

In [24]:
```
data['League Points per game'] = data['League Points'] / data['League Games']
data['UCL Points per game'] = data['UCL Points'] / data['UCL Games']
data['Other int. comp. Points per game'] = data['Other int. comp. Points'] / data['Other int. comp. Games']
data['Other competitions Points per game'] = data['Other competitions Points'] / data['Other competitions Games']
```

Disciplinary points

In [25]:
```
data['League Red cards'] = data['League Red cards'] + data['League Second yellow cards']
data['League Disciplinary points'] = data['League Yellow cards'] + data['League Red cards']*2

data['UCL Red cards'] = data['UCL Red cards'] + data['UCL Second yellow cards']
data['UCL Disciplinary points'] = data['UCL Yellow cards'] + data['UCL Red cards']*2

data['Other int. comp. Red cards'] = (data['Other int. comp. Red cards'] +
                                      data['Other int. comp. Second yellow cards'])

data['Other int. comp. Disciplinary points'] = (data['Other int. comp. Yellow cards'] +
                                                data['Other int. comp. Red cards']*2)

data['Other competitions Red cards'] = (data['Other competitions Red cards'] +
                                        data['Other competitions Second yellow cards'])

data['Other competitions Disciplinary points'] = (data['Other competitions Yellow cards'] +
                                                  data['Other competitions Red cards']*2)
```

Number of substitutions

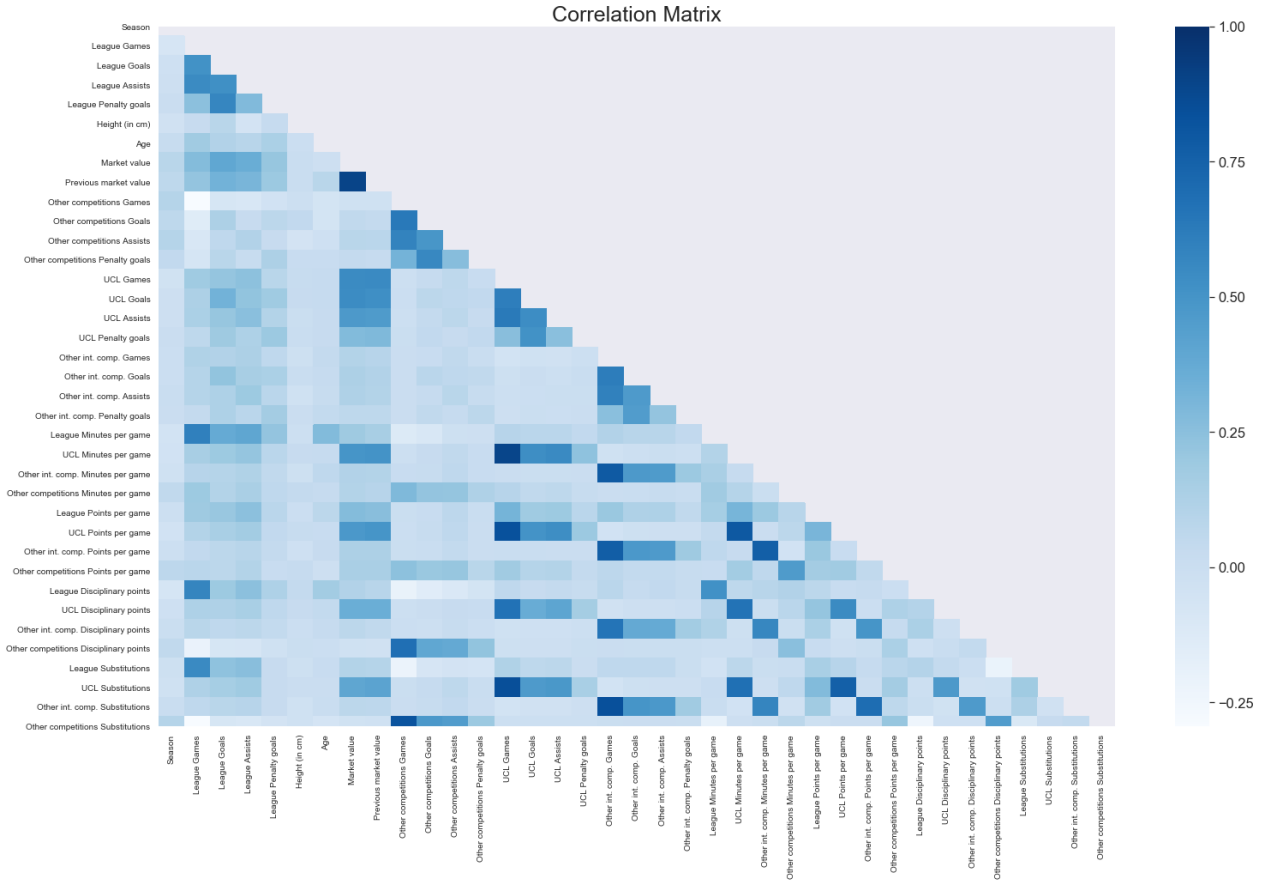
In [26]:
```
data['League Substitutions'] = data['League Subs on'] + data['League Subs off']

data['UCL Substitutions'] = data['UCL Subs on'] + data['UCL Subs off']

data['Other int. comp. Substitutions'] = (data['Other int. comp. Subs on'] +
                                          data['Other int. comp. Subs off'])

data['Other competitions Substitutions'] = (data['Other competitions Subs on'] +
                                            data['Other competitions Subs off'])
```

Drop features used to compute other features

In [27]:
```
data.drop(['League Club games','UCL Club games','Other int. comp. Club games',
           'Other competitions Club games'], axis=1, inplace=True)

data.drop(['League Second yellow cards', 'UCL Second yellow cards',
           'Other int. comp. Second yellow cards',
           'Other competitions Second yellow cards'], axis=1, inplace=True)

data.drop(['League Yellow cards', 'UCL Yellow cards',
           'Other int. comp. Yellow cards',
           'Other competitions Yellow cards'], axis=1, inplace=True)

data.drop(['League Red cards', 'UCL Red cards',
           'Other int. comp. Red cards',
           'Other competitions Red cards'], axis=1, inplace=True)

data.drop(['League Subs on', 'League Subs off', 'UCL Subs on', 'UCL Subs off',
           'Other int. comp. Subs on', 'Other int. comp. Subs off',
           'Other competitions Subs on', 'Other competitions Subs off',], axis=1, inplace=True)

data.drop(['League Own goals', 'UCL Own goals',
           'Other int. comp. Own goals',
           'Other competitions Own goals'], axis=1, inplace=True)

data.drop(['League Points', 'UCL Points',
           'Other int. comp. Points',
           'Other competitions Points'], axis=1, inplace=True)

data.drop(['League Minutes played', 'UCL Minutes played',
           'Other int. comp. Minutes played',
           'Other competitions Minutes played'], axis=1, inplace=True)

data.fillna(0, inplace=True)
```

In [28]:
```
correlations = data.corr()
sns.set(rc={'figure.figsize':(25,15)})
sns.set(font_scale=1.5)

mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

ax = sns.heatmap(correlations, cmap='Blues', mask=mask, yticklabels=correlations.index,
                 xticklabels=correlations.columns)


ax.set_yticklabels(ax.get_yticklabels(), rotation = 0, fontsize = 10)
ax.set_xticklabels(ax.get_xticklabels(), rotation = 90, fontsize = 10)

ax.set_title('Correlation Matrix', fontsize = 25)
```

Out[28]:
```
Text(0.5, 1, 'Correlation Matrix')
```



In [29]:
```
correlations['Market value'].sort_values(ascending=False)
```

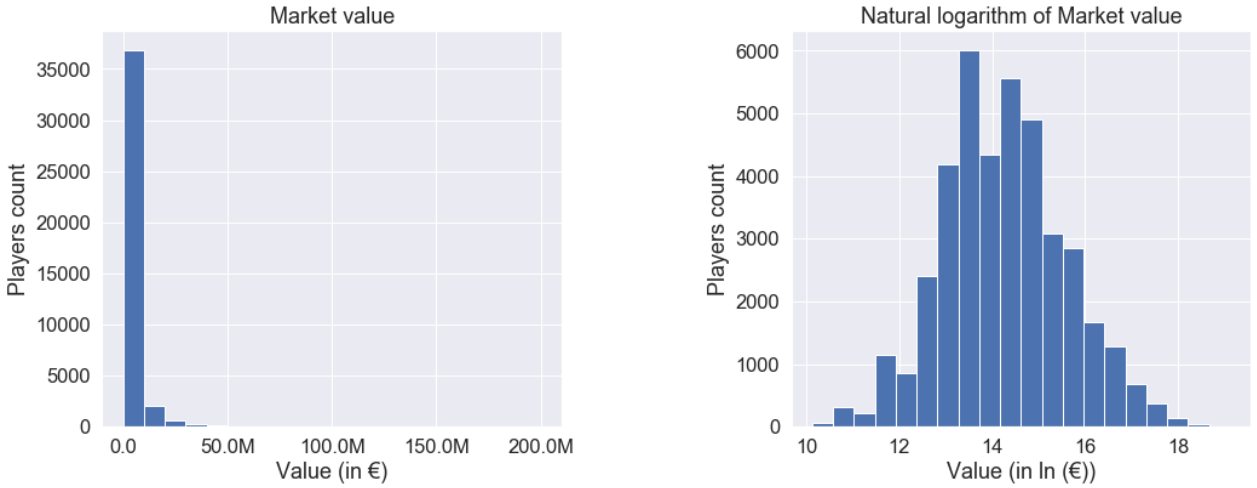
Out[29]:
```
Market value                            1.000
Previous market value                   0.901
UCL Games                               0.548
UCL Goals                               0.541
UCL Minutes per game                    0.495
UCL Points per game                     0.478
UCL Assists                             0.469
UCL Substitutions                       0.402
League Goals                            0.394
League Assists                          0.360
UCL Disciplinary points                 0.353
UCL Penalty goals                       0.281
League Games                            0.276
League Points per game                  0.272
League Penalty goals                    0.212
League Minutes per game                 0.188
Other competitions Points per game      0.145
Other int. comp. Points per game        0.135
Other int. comp. Goals                  0.131
Other int. comp. Assists                0.123
League Disciplinary points              0.115
Other int. comp. Minutes per game       0.114
Other competitions Minutes per game     0.110
Other int. comp. Games                  0.107
League Substitutions                    0.106
Season                                  0.084
Other competitions Assists              0.082
Other int. comp. Substitutions          0.074
Other int. comp. Disciplinary points    0.070
Other int. comp. Penalty goals          0.064
Other competitions Goals                0.052
Other competitions Penalty goals        0.035
Height (in cm)                          0.011
Age                                    -0.014
Other competitions Disciplinary points -0.028
Other competitions Games               -0.029
Other competitions Substitutions       -0.039
Name: Market value, dtype: float64
```

In [30]:
```
(correlations > abs(.7)).sum()
```

Out[30]:
```
Season                                  1
League Games                            1
League Goals                            1
League Assists                          1
League Penalty goals                    1
Height (in cm)                          1
Age                                     1
Market value                            2
Previous market value                   2
Other competitions Games                2
Other competitions Goals                1
Other competitions Assists              1
Other competitions Penalty goals        1
UCL Games                               4
UCL Goals                               1
UCL Assists                             1
UCL Penalty goals                       1
Other int. comp. Games                  4
Other int. comp. Goals                  1
Other int. comp. Assists                1
Other int. comp. Penalty goals          1
League Minutes per game                 1
UCL Minutes per game                    3
Other int. comp. Minutes per game       3
Other competitions Minutes per game     1
League Points per game                  1
UCL Points per game                     4
Other int. comp. Points per game        3
Other competitions Points per game      1
League Disciplinary points              1
UCL Disciplinary points                 1
Other int. comp. Disciplinary points    1
Other competitions Disciplinary points  1
League Substitutions                    1
UCL Substitutions                       3
Other int. comp. Substitutions          2
Other competitions Substitutions        2
dtype: int64
```

Prepare data for Machine Learning algorithms

In [31]:
```
fig, axs = plt.subplots(1, 2, figsize=(17,6), sharey=False)
plt.subplots_adjust(wspace=.5, hspace=.5)


axs[0].hist(data['Market value'], bins=20)
axs[0].set_title('Market value')
axs[0].set_xlabel('Value (in €)')
axs[0].ticklabel_format(axis='both', style='plain')
axs[0].set_ylabel('Players count')

axs[1].hist(np.log(data['Market value']), bins=20)
axs[1].set_title('Natural logarithm of Market value')
axs[1].set_ylabel('Players count')
axs[1].set_xlabel('Value (in ln (€))')

def millions(x, pos):
    if x != 0:
        return '%1.1fM' % (x*1e-6)
    else:
        return x
formatter = FuncFormatter(millions)

axs[0].xaxis.set_major_formatter(formatter)
```



Machine Learning

Create a Traning and Test Set

In [32]:
```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)


# ln(Market value) as label, using previous market value as feature
ln_mkwp_x_train = train_set.drop(['Market value'], axis=1).copy()
ln_mkwp_y_train = np.log(train_set['Market value'].copy())

ln_mkwp_x_test = test_set.drop(['Market value'], axis=1).copy()
ln_mkwp_y_test = np.log(test_set['Market value'].copy())


# ln(Market value) as label, not using previous market value as feature
ln_mknp_x_train = train_set.drop(['Market value', 'Previous market value'], axis=1).copy()
ln_mknp_y_train = np.log(train_set['Market value'].copy())

ln_mknp_x_test = test_set.drop(['Market value', 'Previous market value'], axis=1).copy()
ln_mknp_y_test = np.log(test_set['Market value'].copy())
```

Handling Categorical Attributes

In [34]:
```
# ln(Market value) as label, using previous market value as feature
ln_mkwp = data.copy().drop('Market value', axis=1)
ln_mkwp_cat = ln_mkwp[['Nationality', 'Position', 'Main position', 'Foot', 'Player agent',
                       'Outfitter', 'League', 'Club', 'Played UCL', 'Played Other int. comp.']]


ln_mkwp_num = ln_mkwp.drop(['Nationality', 'Position', 'Main position', 'Foot', 'Player agent',
                        'Outfitter', 'League', 'Club', 'Played UCL', 'Played Other int. comp.'], axis=1)


# ln(Market value) as label, not using previous market value as feature
ln_mknp = data.copy().drop('Market value', axis=1)
ln_mknp_cat = ln_mknp[['Nationality', 'Position', 'Main position', 'Foot', 'Player agent',
                       'Outfitter', 'League', 'Club', 'Played UCL', 'Played Other int. comp.']]


ln_mknp_num = ln_mknp.drop(['Nationality', 'Position', 'Main position', 'Foot', 'Player agent',
                            'Outfitter', 'League', 'Club', 'Played UCL', 'Played Other int. comp.',
                            'Previous market value'], axis=1)
```

Pipeline - Categorical encoder

In [35]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# Create a class to select numerical or categorical columns
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values


def num_pipeline(num_attribs):
    return Pipeline([
        ('selector', DataFrameSelector(list(num_attribs)))
         ])

def cat_pipeline(cat_attribs):
    return Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(sparse=False))
    ])
```

In [36]:
```
# ln(Market value) as label, using previous market value as feature
ln_mkwp_pipeline = ColumnTransformer([
        ('num', num_pipeline(ln_mkwp_num), list(ln_mkwp_num)),
        ('cat', OneHotEncoder(sparse=False), list(ln_mkwp_cat)),
    ])

ln_mkwp_x_train_prepared = ln_mkwp_pipeline.fit_transform(ln_mkwp_x_train)
ln_mkwp_x_test_prepared = ln_mkwp_pipeline.fit_transform(ln_mkwp_x_test)


# ln(Market value) as label, not using previous market value as feature
ln_mknp_pipeline = ColumnTransformer([
        ('num', num_pipeline(ln_mknp_num), list(ln_mknp_num)),
        ('cat', OneHotEncoder(sparse=False), list(ln_mknp_cat)),
    ])

ln_mknp_x_train_prepared = ln_mknp_pipeline.fit_transform(ln_mknp_x_train)
ln_mknp_x_test_prepared = ln_mknp_pipeline.fit_transform(ln_mknp_x_test)
```

In [37]:
```
print('If "Previous market value" is known:')
print(f'Before encoding categorical variables:',
      f'\nNumber of records = {ln_mkwp_x_train.shape[0] + ln_mkwp_x_test.shape[0]}',
      f'\nNumber of features = {ln_mkwp_x_train.shape[1]}')

print(f'\nAfter encoding categorical variables:',
      f'\nNumber of records = {ln_mkwp_x_train_prepared.shape[0] + ln_mkwp_x_test_prepared.shape[0]}',
      f'\nNumber of features = {ln_mkwp_x_train_prepared.shape[1]}\n\n')

print('If "Previous market value" is not known:')
print(f'Before encoding categorical variables:',
      f'\nNumber of records = {ln_mknp_x_train.shape[0] + ln_mknp_x_test.shape[0]}',
      f'\nNumber of features = {ln_mknp_x_train.shape[1]}')

print(f'\nAfter encoding categorical variables:',
      f'\nNumber of records = {ln_mknp_x_train_prepared.shape[0] + ln_mknp_x_test_prepared.shape[0]}',
      f'\nNumber of features = {ln_mknp_x_train_prepared.shape[1]}')
```

```
If "Previous market value" is known:
Before encoding categorical variables:
Number of records = 40173
Number of features = 46

After encoding categorical variables:
Number of records = 40173
Number of features = 111


If "Previous market value" is not known:
Before encoding categorical variables:
Number of records = 40173
Number of features = 45

After encoding categorical variables:
Number of records = 40173
Number of features = 110
```

Training and Evaluating on the Training Set

Ordinary least squares Linear Regression

In [38]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score as R2


#ln(Market value) as label, using previous market value as feature
ln_mkwp_lin_reg = LinearRegression(fit_intercept=True, n_jobs=-1)
ln_mkwp_lin_reg.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_train_lin_predictions = ln_mkwp_lin_reg.predict(ln_mkwp_x_train_prepared)
ln_mkwp_lin_mse = MSE(ln_mkwp_y_train, ln_mkwp_train_lin_predictions)
ln_mkwp_lin_rmse = np.sqrt(ln_mkwp_lin_mse)
ln_mkwp_lin_r2 = R2(ln_mkwp_y_train, ln_mkwp_train_lin_predictions)

print('ln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mkwp_lin_rmse, '\nLNMKWP R2 =', ln_mkwp_lin_r2)



#ln(Market value) as label, not using previous market value as feature
ln_mknp_lin_reg = LinearRegression(fit_intercept=True, n_jobs=-1)
ln_mknp_lin_reg.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_train_lin_predictions = ln_mknp_lin_reg.predict(ln_mknp_x_train_prepared)
ln_mknp_lin_mse = MSE(ln_mknp_y_train, ln_mknp_train_lin_predictions)
ln_mknp_lin_rmse = np.sqrt(ln_mknp_lin_mse)
ln_mknp_lin_r2 = R2(ln_mknp_y_train, ln_mknp_train_lin_predictions)

print('\n\nln(Market value) as label, not using previous market value as feature',
      '\nLNMKNP RMSE =', ln_mknp_lin_rmse, '\nLNMKNP R2 =', ln_mknp_lin_r2)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.6962617343485532
LNMKWP R2 = 0.734262006215131


ln(Market value) as label, not using previous market value as feature
LNMKNP RMSE = 0.7499200606145551
LNMKNP R2 = 0.691724834654293
```

Ridge

In [40]:
```
from sklearn.linear_model import Ridge

#ln(Market value) as label, using previous market value as feature
ln_mkwp_rid_reg = Ridge(random_state=42)
ln_mkwp_rid_reg.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_rid_predictions = ln_mkwp_rid_reg.predict(ln_mkwp_x_train_prepared)
ln_mkwp_rid_mse = MSE(ln_mkwp_y_train, ln_mkwp_rid_predictions)
ln_mkwp_rid_rmse = np.sqrt(ln_mkwp_rid_mse)
ln_mkwp_rid_r2 = R2(ln_mkwp_y_train, ln_mkwp_rid_predictions)

print('ln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mkwp_rid_rmse, '\nLNMKWP R2 =', ln_mkwp_rid_r2)



#ln(Market value) as label, not using previous market value as feature
ln_mknp_rid_reg = Ridge(random_state=42)
ln_mknp_rid_reg.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_rid_predictions = ln_mknp_rid_reg.predict(ln_mknp_x_train_prepared)
ln_mknp_rid_mse = MSE(ln_mknp_y_train, ln_mknp_rid_predictions)
ln_mknp_rid_rmse = np.sqrt(ln_mknp_rid_mse)
ln_mknp_rid_r2 = R2(ln_mknp_y_train, ln_mknp_rid_predictions)

print('\n\nln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mknp_rid_rmse, '\nLNMKWP R2 =', ln_mknp_rid_r2)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.6962620409366952
LNMKWP R2 = 0.7342617721878056


ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.7499204066552911
LNMKWP R2 = 0.6917245501551967

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:148: LinAlgWarning: Ill-conditioned matrix
(rcond=5.95093e-19): result may not be accurate.
  overwrite_a=True).T
```

Lasso

In [41]:
```
from sklearn.linear_model import Lasso

#ln(Market value) as label, using previous market value as feature
ln_mkwp_las_reg = Lasso(random_state=42)
ln_mkwp_las_reg.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_las_predictions = ln_mkwp_las_reg.predict(ln_mkwp_x_train_prepared)
ln_mkwp_las_mse = MSE(ln_mkwp_y_train, ln_mkwp_las_predictions)
ln_mkwp_las_rmse = np.sqrt(ln_mkwp_las_mse)
ln_mkwp_las_r2 = R2(ln_mkwp_y_train, ln_mkwp_las_predictions)

print('ln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mkwp_las_rmse, '\nLNMKWP R2 =', ln_mkwp_las_r2)



#ln(Market value) as label, not using previous market value as feature
ln_mknp_las_reg = Lasso(random_state=42)
ln_mknp_las_reg.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_las_predictions = ln_mknp_las_reg.predict(ln_mknp_x_train_prepared)
ln_mknp_las_mse = MSE(ln_mknp_y_train, ln_mknp_las_predictions)
ln_mknp_las_rmse = np.sqrt(ln_mknp_las_mse)
ln_mknp_las_r2 = R2(ln_mknp_y_train, ln_mknp_las_predictions)

print('\n\nln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mknp_las_rmse, '\nLNMKWP R2 =', ln_mknp_las_r2)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.9298130751035072
LNMKWP R2 = 0.5260856112290235


ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 1.07364970194343
LNMKWP R2 = 0.3681211594901228
```

CART Decision Tree

In [43]:
```
from sklearn import tree


#Ln(Market value) as label, using previous market value as feature
ln_mkwp_tree = tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=10, random_state=42)
ln_mkwp_tree.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_train_tree_predictions = ln_mkwp_tree.predict(ln_mkwp_x_train_prepared)
ln_mkwp_tree_mse = MSE(ln_mkwp_y_train, ln_mkwp_train_tree_predictions)
ln_mkwp_tree_rmse = np.sqrt(ln_mkwp_tree_mse)
ln_mkwp_tree_r2 = R2(ln_mkwp_y_train, ln_mkwp_train_tree_predictions)

print('ln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mkwp_tree_rmse, '\nLNMKWP R2 =', ln_mkwp_tree_r2)




#Ln(Market value) as label, not using previous market value as feature
ln_mknp_tree = tree.DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=10, random_state=42)
ln_mknp_tree.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_train_tree_predictions = ln_mknp_tree.predict(ln_mknp_x_train_prepared)
ln_mknp_tree_mse = MSE(ln_mknp_y_train, ln_mknp_train_tree_predictions)
ln_mknp_tree_rmse = np.sqrt(ln_mknp_tree_mse)
ln_mknp_tree_r2 = R2(ln_mknp_y_train, ln_mknp_train_tree_predictions)

print('\n\nln(Market value) as label, not using previous market value as feature',
      '\nLNMKNP RMSE =', ln_mknp_tree_rmse, '\nLNMKNP R2 =', ln_mknp_tree_r2)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.45934642552104377
LNMKWP R2 = 0.8843383777935785


ln(Market value) as label, not using previous market value as feature
LNMKNP RMSE = 0.7983320649414579
LNMKNP R2 = 0.6506379374829665
```

Ensamble methods

Random Forrest

In [47]:
```
from sklearn.ensemble import RandomForestRegressor


#ln(Market value) as label, using previous market value as feature
ln_mkwp_forest = RandomForestRegressor(n_estimators=200, criterion='mse', max_depth=10, oob_score=True,
                                       random_state=42, n_jobs=4)
ln_mkwp_forest.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_train_forest_predictions = ln_mkwp_forest.predict(ln_mkwp_x_train_prepared)
ln_mkwp_forest_mse = MSE(ln_mkwp_y_train, ln_mkwp_train_forest_predictions)
ln_mkwp_forest_rmse = np.sqrt(ln_mkwp_forest_mse)
ln_mkwp_forest_r2 = R2(ln_mkwp_y_train, ln_mkwp_train_forest_predictions)

print('ln(Market value) as label, using previous market value as feature',
      '\nLNMKWP RMSE =', ln_mkwp_forest_rmse, '\nLNMKWP R2 =', ln_mkwp_forest_r2)



#ln(Market value) as label, not using previous market value as feature
ln_mknp_forest = RandomForestRegressor(n_estimators=200, criterion='mse', max_depth=10, oob_score=True,
                                       random_state=42, n_jobs=4)
ln_mknp_forest.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_train_forest_predictions = ln_mknp_forest.predict(ln_mknp_x_train_prepared)
ln_mknp_forest_mse = MSE(ln_mknp_y_train, ln_mknp_train_forest_predictions)
ln_mknp_forest_rmse = np.sqrt(ln_mknp_forest_mse)
ln_mknp_forest_r2 = R2(ln_mknp_y_train, ln_mknp_train_forest_predictions)

print('\n\nln(Market value) as label, not using previous market value as feature',
      '\nLNMKNP RMSE =', ln_mknp_forest_rmse, '\nLNMKNP R2 =', ln_mknp_forest_r2)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.4162896656762193
LNMKWP R2 = 0.9050051948522376


ln(Market value) as label, not using previous market value as feature
LNMKNP RMSE = 0.7379027191300592
LNMKNP R2 = 0.7015257853142046
```

Model Comparison - K-folds validation

In [48]:
```
from sklearn.model_selection import cross_val_score
```

In [51]:
```
#ln(Market value) as label, using previous market value as feature
ln_mkwp_lin_reg_scores = cross_val_score(ln_mkwp_lin_reg, ln_mkwp_x_train_prepared, ln_mkwp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)

ln_mkwp_tree_scores = cross_val_score(ln_mkwp_tree, ln_mkwp_x_train_prepared, ln_mkwp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)

ln_mkwp_forest_scores = cross_val_score(ln_mkwp_forest, ln_mkwp_x_train_prepared, ln_mkwp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)


ln_mkwp_lin_reg_cv_mean_rmse = np.sqrt(-ln_mkwp_lin_reg_scores).mean()
ln_mkwp_tree_cv_mean_rmse = np.sqrt(-ln_mkwp_tree_scores).mean()
ln_mkwp_forest_cv_mean_rmse = np.sqrt(-ln_mkwp_forest_scores).mean()

ln_mkwp_lin_reg_cv_std_rmse = np.sqrt(-ln_mkwp_lin_reg_scores).std()
ln_mkwp_tree_cv_std_rmse = np.sqrt(-ln_mkwp_tree_scores).std()
ln_mkwp_forest_cv_std_rmse = np.sqrt(-ln_mkwp_forest_scores).std()


ln_mkwp_cv_metrics = {'Algorithm' : ['OLS', 'CART', 'Random Forest'],
                      'Mean RMSE': [ln_mkwp_lin_reg_cv_mean_rmse,
                                    ln_mkwp_tree_cv_mean_rmse,
                                    ln_mkwp_forest_cv_mean_rmse],
                      'RMSE Standard deviation':[ln_mkwp_lin_reg_cv_std_rmse,
                                                 ln_mkwp_tree_cv_std_rmse,
                                                 ln_mkwp_forest_cv_std_rmse]}

pd.DataFrame(ln_mkwp_cv_metrics)
```

Out[51]:

	Algorithm	Mean RMSE	RMSE Standard deviation
0	OLS	0.699	0.017
1	CART	0.557	0.006
2	Random Forest	0.489	0.009

In [53]:
```
#Ln(Market value) as label, not using previous market value as feature
ln_mknp_lin_reg_scores = cross_val_score(ln_mknp_lin_reg, ln_mknp_x_train_prepared, ln_mknp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)

ln_mknp_tree_scores = cross_val_score(ln_mknp_tree, ln_mknp_x_train_prepared, ln_mknp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)

ln_mknp_forest_scores = cross_val_score(ln_mknp_forest, ln_mknp_x_train_prepared, ln_mknp_y_train.values,
                                scoring=('neg_mean_squared_error'), cv=10, n_jobs=4)


ln_mknp_lin_reg_cv_mean_rmse = np.sqrt(-ln_mknp_lin_reg_scores).mean()
ln_mknp_tree_cv_mean_rmse = np.sqrt(-ln_mknp_tree_scores).mean()
ln_mknp_forest_cv_mean_rmse = np.sqrt(-ln_mknp_forest_scores).mean()

ln_mknp_lin_reg_cv_std_rmse = np.sqrt(-ln_mknp_lin_reg_scores).std()
ln_mknp_tree_cv_std_rmse = np.sqrt(-ln_mknp_tree_scores).std()
ln_mknp_forest_cv_std_rmse = np.sqrt(-ln_mknp_forest_scores).std()

ln_mknp_cv_metrics = {'Algorithm' : ['OLS', 'CART', 'Random Forest'],
                      'Mean RMSE': [ln_mknp_lin_reg_cv_mean_rmse,
                                    ln_mknp_tree_cv_mean_rmse,
                                    ln_mknp_forest_cv_mean_rmse],
                      'RMSE Standard deviation':[ln_mknp_lin_reg_cv_std_rmse,
                                                 ln_mknp_tree_cv_std_rmse,
                                                 ln_mknp_forest_cv_std_rmse]}

pd.DataFrame(ln_mknp_cv_metrics)
```

Out[53]:

	Algorithm	Mean RMSE	RMSE Standard deviation
0	OLS	0.752	0.012
1	CART	0.880	0.012
2	Random Forest	0.794	0.009

Tunning models - Grid Search

Random Forest

In [56]:
```
# Create the random grid
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'max_features': [10, 50, 'auto'],
     'n_estimators': [100, 200, 500]}
  ]

forest_reg = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=10,
                           scoring='neg_mean_squared_error', return_train_score=True, n_jobs=4)
```

In [58]:
```
#Ln(Market value) as label, using previous market value as feature
ln_mkwp_gr=grid_search.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)

ln_mkwp_gr.best_estimator_
```

Out[58]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=50, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [60]:
```
#Ln(Market value) as label, not using previous market value as feature
ln_mknp_gr=grid_search.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)

ln_mknp_gr.best_estimator_
```

Out[60]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=50, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

Predicting on the Test set

If Previous Market Value is known

In [61]:
```
ln_mkwp_final_forest = RandomForestRegressor(n_estimators=500, min_samples_leaf=1, criterion='mse',
                                             random_state=42, n_jobs=4)

ln_mkwp_final_forest.fit(ln_mkwp_x_train_prepared, ln_mkwp_y_train.values)
```

Out[61]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=4, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

In [66]:
```
#ln(Market value) as label, using previous market value as feature

ln_mkwp_test_forest_predictions = ln_mkwp_final_forest.predict(ln_mkwp_x_test_prepared)
ln_mkwp_test_forest_mse = MSE(ln_mkwp_y_test, ln_mkwp_test_forest_predictions)
ln_mkwp_test_forest_rmse = np.sqrt(ln_mkwp_test_forest_mse)
ln_mkwp_test_forest_r2 = R2(ln_mkwp_y_test, ln_mkwp_test_forest_predictions)


confidence = 0.95
squared_errors = (ln_mkwp_test_forest_predictions - ln_mkwp_y_test) ** 2
mean = squared_errors.mean()
m = len(squared_errors)

ln_mkwp_test_forest_rmse_95_ci=np.sqrt(stats.t.interval(confidence, m - 1,
                         loc=np.mean(squared_errors),
                         scale=stats.sem(squared_errors)))


print('ln(Market value) as label, using previous market value as feature\nLNMKWP RMSE =',
      ln_mkwp_test_forest_rmse, '\nLNMKWP R2 =', ln_mkwp_test_forest_r2,
      '\nLNMKWP RMSE 95% CI =', ln_mkwp_test_forest_rmse_95_ci)
```

```
ln(Market value) as label, using previous market value as feature
LNMKWP RMSE = 0.4574377757817017
LNMKWP R2 = 0.8806409814271537
LNMKWP RMSE 95% CI = [0.44533357 0.46922984]
```

Top 10 most important features

In [67]:
```
ln_mkwp_features_importance = ln_mkwp_final_forest.feature_importances_
cat_encoder = ln_mkwp_pipeline.named_transformers_['cat']
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = list(ln_mkwp_num) + cat_one_hot_attribs
sorted(zip(ln_mkwp_features_importance, attributes), reverse=True)[:10]
```

Out[67]:
```
[(0.7837923403242727, 'Previous market value'),
 (0.03054948702622499, 'Age'),
 (0.02841507539912998, 'League Games'),
 (0.013697508328784695, 'League Minutes per game'),
 (0.011974167596970432, 'League Points per game'),
 (0.011124945258368395, 'Season'),
 (0.010683038809960344, 'League Goals'),
 (0.006561625214384067, 'Other competitions Minutes per game'),
 (0.006094613397068786, 'Other competitions Games'),
 (0.005804345064534985, 'Height (in cm)')]
```

Predictions

In [68]:
```
ln_mkwp_df = pd.DataFrame({'Actual':np.e**ln_mkwp_y_test, 'Predicted':np.e**ln_mkwp_test_forest_predictions})
ln_mkwp_df
```

Out[68]:

	Actual	Predicted
28900	600000.000	485744.894
24209	8000000.000	5282936.099
11240	1000000.000	957686.324
31834	1200000.000	972272.301
29904	1800000.000	1699676.311
...	...	...
30684	5000000.000	5664167.192
1697	1300000.000	1365552.884
36303	400000.000	450976.685
32760	250000.000	272203.099
666	2000000.000	1557287.715

8035 rows × 2 columns

If Previous Market Value is not known

In [69]:
```
ln_mknp_final_forest = RandomForestRegressor(n_estimators=500, min_samples_leaf=1, criterion='mse',
                                             max_depth=50, oob_score=True, random_state=42, n_jobs=4)

ln_mknp_final_forest.fit(ln_mknp_x_train_prepared, ln_mknp_y_train.values)
```

Out[69]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=50, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=4, oob_score=True,
                      random_state=42, verbose=0, warm_start=False)
```

In [70]:
```
#ln(Market value) as label, not using previous market value as feature

ln_mknp_test_forest_predictions = ln_mknp_final_forest.predict(ln_mknp_x_test_prepared)
ln_mknp_test_forest_mse = MSE(ln_mknp_y_test, ln_mknp_test_forest_predictions)
ln_mknp_test_forest_rmse = np.sqrt(ln_mknp_test_forest_mse)
ln_mknp_test_forest_r2 = R2(ln_mknp_y_test, ln_mknp_test_forest_predictions)

confidence = 0.95
squared_errors = (ln_mknp_test_forest_predictions - ln_mknp_y_test) ** 2
mean = squared_errors.mean()
m = len(squared_errors)

ln_mknp_test_forest_rmse_95_ci=np.sqrt(stats.t.interval(confidence, m - 1,
                         loc=np.mean(squared_errors),
                         scale=stats.sem(squared_errors)))

print('\n\nln(Market value) as label, not using previous market value as feature\nLNMKNP RMSE =',
      ln_mknp_test_forest_rmse, '\nLNMKNP R2 =', ln_mknp_test_forest_r2,
     '\nLNMKNP RMSE 95% CI =', ln_mknp_test_forest_rmse_95_ci)
```

```
ln(Market value) as label, not using previous market value as feature
LNMKNP RMSE = 0.6750880457858243
LNMKNP R2 = 0.7400367115820219
LNMKNP RMSE 95% CI = [0.66145432 0.68845183]
```

Top 10 most important features

In [71]:
```
ln_mknp_features_importance = ln_mknp_final_forest.feature_importances_
cat_encoder = ln_mknp_pipeline.named_transformers_['cat']
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = list(ln_mknp_num) + cat_one_hot_attribs
sorted(zip(ln_mknp_features_importance, attributes), reverse=True)[:10]
```

Out[71]:
```
[(0.17155067116365044, 'UCL Minutes per game'),
 (0.12630553355672683, 'League Games'),
 (0.04965448745410156, 'Age'),
 (0.044453061651216194, 'League Points per game'),
 (0.04324527412514572, 'League Goals'),
 (0.0405866501076086, 'League Minutes per game'),
 (0.03375758028532208, 'Season'),
 (0.024008378369048698, 'Other int. comp. Minutes per game'),
 (0.017971521519330413, 'UCL Games'),
 (0.017757981922154616, 'Other int. comp. Games')]
```

Predictions

In [72]:
```
ln_mknp_df = pd.DataFrame({'Actual':np.e**ln_mknp_y_test, 'Predicted':np.e**ln_mknp_test_forest_predictions})
ln_mknp_df
```

Out[72]:

	Actual	Predicted
28900	600000.000	613888.756
24209	8000000.000	4517585.980
11240	1000000.000	974899.788
31834	1200000.000	855351.541
29904	1800000.000	1140403.650
...	...	...
30684	5000000.000	7048042.432
1697	1300000.000	2074682.291
36303	400000.000	394523.984
32760	250000.000	310746.997
666	2000000.000	481240.219

8035 rows × 2 columns

Comparison

In [87]:
```
compare_df = pd.DataFrame({'Actual Market Value':np.e**ln_mknp_y_test,
                           'RF knowing previous market value':np.e**ln_mkwp_test_forest_predictions,
                           'RF not knowing previous market value':np.e**ln_mknp_test_forest_predictions})

comparison_sample_with_replacement = compare_df.sample(n=20,replace=True)
comparison_sample_with_replacement.reset_index(drop=True, inplace=True)
comparison_sample_with_replacement.index += 1
comparison_sample_with_replacement
```

Out[87]:

	Actual Market Value	RF knowing previous market value	RF not knowing previous market value
1	2500000.000	2744772.777	3424234.416
2	450000.000	534559.762	640662.102
3	450000.000	727379.081	883668.835
4	450000.000	656016.565	542905.470
5	500000.000	816656.472	685111.340
6	2000000.000	2679369.770	2393770.951
7	1000000.000	1284131.771	1999046.151
8	1800000.000	1477454.638	1682112.739
9	800000.000	834008.054	661585.083
10	250000.000	289911.555	332323.427
11	2000000.000	1781684.955	914659.095
12	3000000.000	2889861.170	3966584.499
13	3500000.000	3645898.281	5358250.090
14	800000.000	1493403.729	1546552.634
15	9900000.000	13980638.575	10679951.114
16	200000.000	184174.010	353084.715
17	750000.000	1267958.178	904850.128
18	9000000.000	11785935.114	5292499.727
19	100000.000	210671.273	322288.348
20	6000000.000	3452539.969	4035642.993

In [94]:
```
fig1 = plt.figure()
ax1 = fig1.add_subplot(1, 1, 1)

comparison_sample_with_replacement.plot(kind='bar', figsize=(16,10), ax=ax1, width=.9)

ax1.set_title('Comparison between actual and predicted market values\n(Random sample of 20 players)',
              pad=20, fontfamily='Times New Roman', fontsize=25,
              horizontalalignment='center')
ax1.set_xlabel('Players', labelpad=20, fontfamily='Times New Roman', fontsize=25)
ax1.set_ylabel('Market Value in €', labelpad=20, fontfamily='Times New Roman', fontsize=25)
ax1.legend(loc='best')
ax1.tick_params(axis='both', labelsize=15, rotation=0)
ax1.ticklabel_format(axis='y', style='plain')


def millions(x, pos):
    if x != 0:
        return '€%1.1fM' % (x*1e-6)
    else:
        return x

formatter = FuncFormatter(millions)
ax1.yaxis.set_major_formatter(formatter)
ax1.grid(False, axis='x')
plt.yticks(fontname = "Times New Roman")
plt.xticks(fontname = "Times New Roman")
plt.legend(prop={"size":18, 'family':"Times New Roman"})
```

Out[94]: <matplotlib.legend.Legend at 0x19fd8adc048>