HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION TECHNOLOGY AND COMMUNICATION

---○🕮○---

MACHINE LEARNING

Project name: Prediction of football player’s price

Asistant Lecturer: Assoc.Prof Than Quang Khoat

Student name:

Nguyen Nho Trung – 20204894

Ho Minh Khoi – 20204917

Luong Quang Binh – 20200068

Group: 14

Class: DS & AI

I. Introduction:

Our project is applying Machine Learning to predict the price of football players. In detail, we will predict the price of players who are playing in the top 5 leagues in Europe, including Premier League, La Liga, Bundesliga, Serie A and Ligue 1.

The transfer market in Europe takes place every year, there are very successful signings but there are failed ones. We can mention like Harry Maguire, Eden Hazard, Jack Grealish, ... These players are very high prices but play very badly. So, evaluation of player prices is very important for team managers, especially players they want to buy for their club. This project aims to help managers estimate player prices based on player statistics and their performance in the 2021-2022 season.

II. Dataset:

- The dataset for this project consists of 2 datasets.
- First dataset

+ It has many attributes of the football statistics of the players. It includes name, overall, position, potential, preferred_foot, value, wage, club, best_position, age, volleys, vision, strength, standing_tackle, stamina, Sprint_Speed, Sliding_Tackle, Skill_Moves, Shot_Power, Short_Passing, Reactions, Positioning, Penalties, Marking, Long_Shots, Long_Passing, Jumping, International_Reputation, Interceptions, Heading_Accuracy, GK_Reflexes, GK_Positioning, GK_Kicking, GK_Handling, GK_Diving, Finishing, FK_Accuracy, Dribbling, Curve, Crossing, Composure, Ball_Control, Balance, Agility, Aggression, Acceleration.

+ This dataset crawled from sofifa website using Scrapy.
(Link website: https://sofifa.com/)

+ The name of dataset in csv file: player_index.csv

- Second dataset:

+ It has many statistics about the performance of the players in the season 2021-2022. But we are only interested in some statistics that are important for our player price prediction problem, such as: Players (Player's name), MP (Matches played), Min(Minutes played), Goals (number of goals divided by number of matches played), Assists(number of assists divided by number of matches played), CrdY(number of yellow card divided by number of matches played), CrdR (number of red cards divided by number of matches played), Comp (name of league), Shots (the total of shots divided by number of matches played).

+ This dataset is available on Kaggle, link dataset: https://www.kaggle.com/datasets/vivovinco/20212022-football-player-stats

+ The name of data in csv file: 2021-2022 Football Player Stats.csv

III. Integration data and Data cleaning:

- Firstly, we removed duplicated records in 2 datasets
  - ⇨ 136 duplicated records removed in 2021-2022 Football Player Stats.csv and 2 duplicated records removed in player_index.csv
- Second, join 2 tables based on player's name in 2 tables, we have the idea:

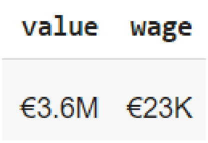
  + 2 full names taken from 2 tables are the same if: we have at least 2 similar words in any 2 full names of 2 tables, then those 2 full names are full name of 1 player.

  + After we have found matching pairs of full names, we will select the columns/attributes that are relevant to our prediction problem, and next, we use pd.merge to join the two tables with together.
- Third, we check missing values: the dataset don't have missing values.
- Fourth, we will deal with position and best_position columns:

  + we replace 'SUB' and 'RES' position by best_position (SUB: substitution player, RES: young players who haven't played much but are on the first team)

+ we obtain LCB, CDM, LB, LM, LDM, RB, RCB, LS, LCM, RCM, RM, LWB, RDM, ST, CM, RW, RS, GK, CAM, RWB, RF, LF, CB, LAM, CF, LW, RAM positions. So we will convert these position into 4 positions in common, including: goalkeeper, defender, midfielder, striker.

- Fifth, we will process with value and wage columns: because these columns contain special characters
  - ⇨ we need to convert it into numerical.

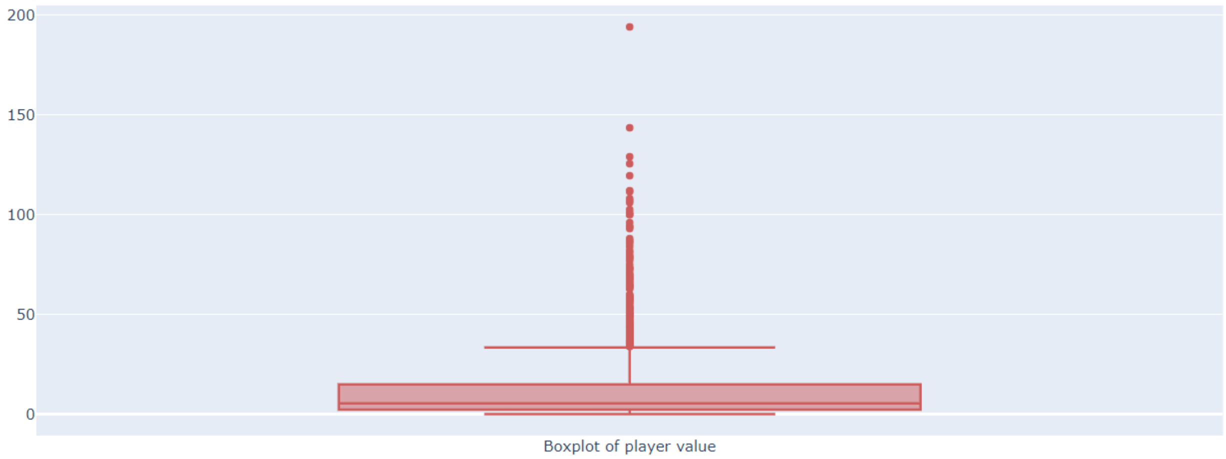


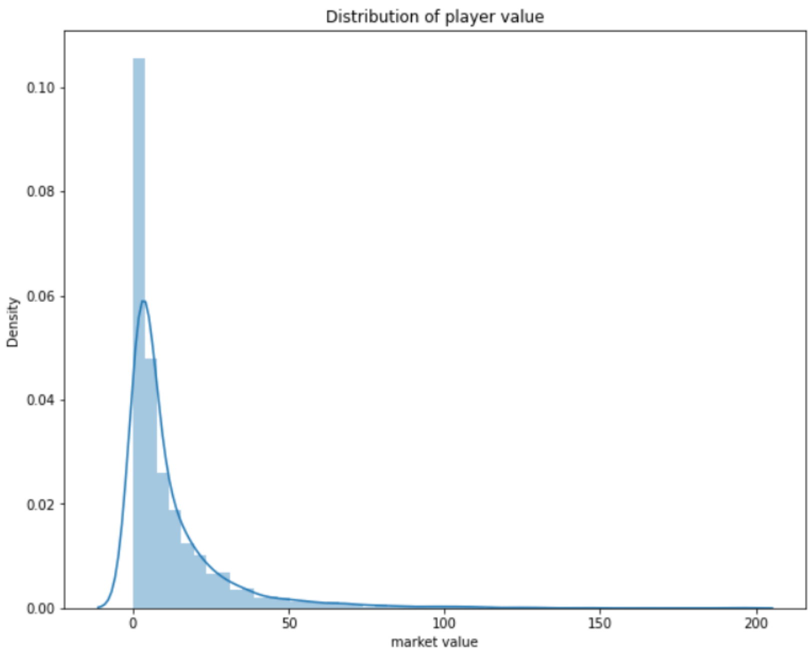
value	wage
€3.6M	€23K

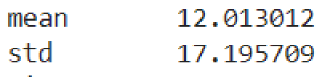
- Sixth, we removed redundant columns that are not relevant in the predictive model and import datafram into csv file, namely ‘processed data.csv’. This data set have 2002 records

IV. Exploratory data analysis:

1. See distribution and boxplot of market value column:



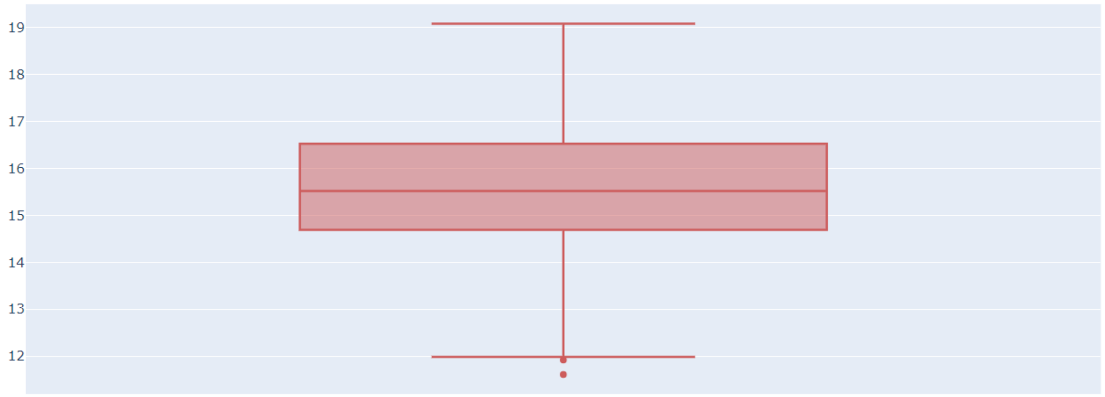



```
mean        12.013012
std         17.195709
```

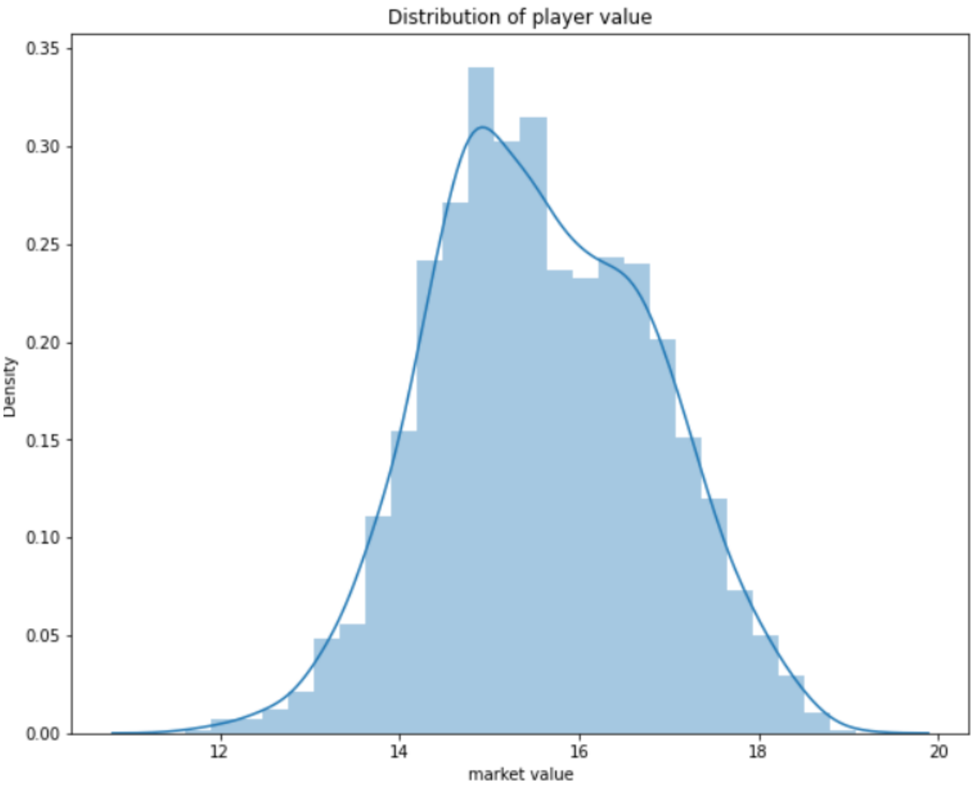
We can see that: the distribution of player value is very positive skewed. It makes predicting high priced players very difficult. The highest value player is €194M and the lowest value player is just €110000. The mean is approximately € 12M but with a very large standard deviation of €17M. Moreover, we see that the data has many outliers because most of the players are of low value.

=> we decided to use log transformation from target variable to reduce variance in this column.

After we use log transformation technique, we have :



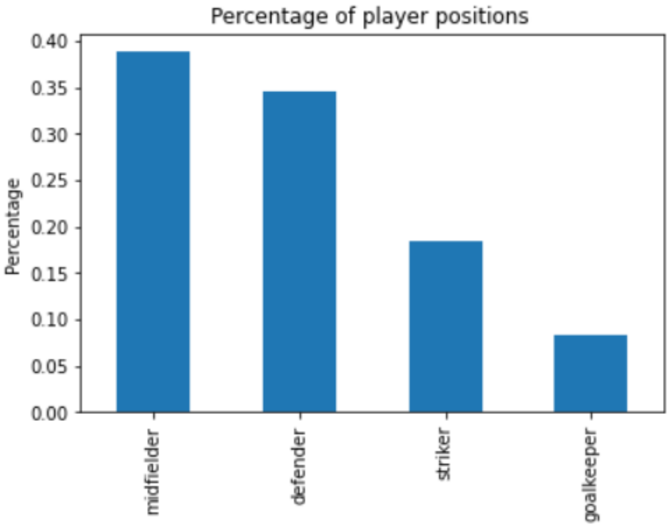
Boxplot of player value



You can see that: Market log value variable is more normally distributed and has a much lower standard deviation compared to the mean. With this technique, we only 2 outliers in market value variable.

So, we decided to training model with market log value as a target variable.

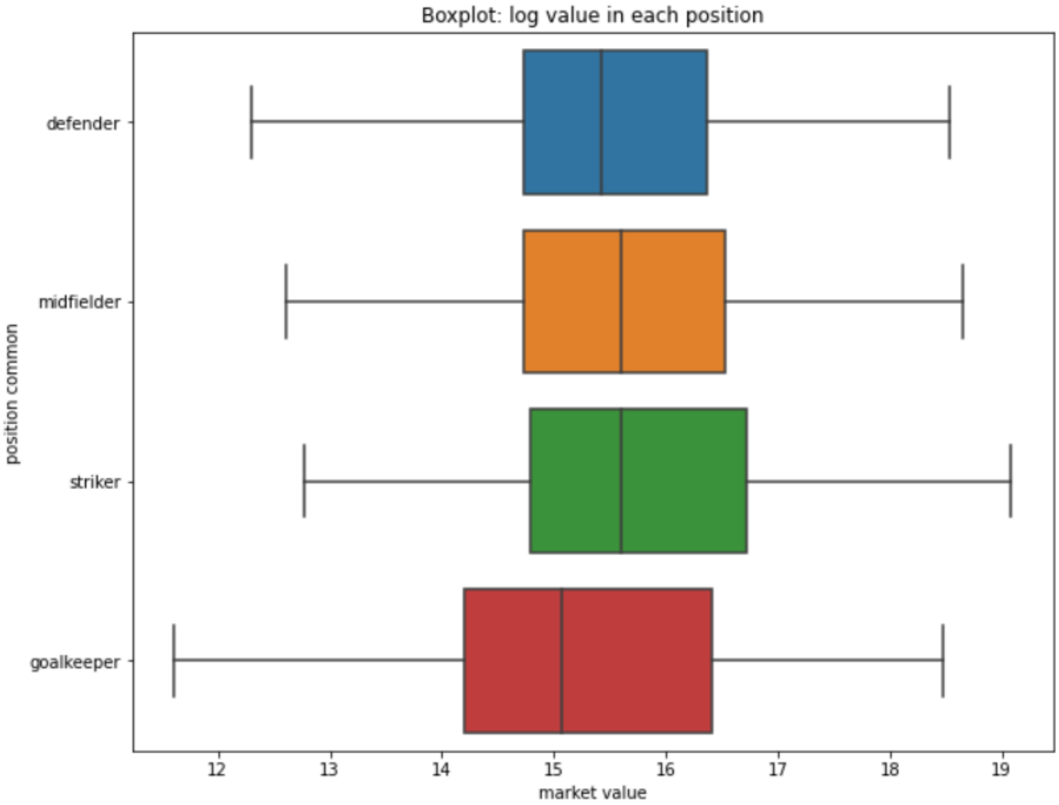
2. Position:



- The percentage of player positions:

The percentage of players playing in the midfield position is the highest with approximately 38%. This is also quite understandable because the teams in the top 5 European leagues have a lot of different styles of play depending on the game, including counter-attack, ball control, ... Therefore, they need many midfielders. There is different gameplay to deploy different tactics. Meanwhile, the goalkeeper is the least with about 8%, because each team usually has only 1 main goalkeeper and 2-3 reserve goalkeepers.

- Next , we see log value in each position

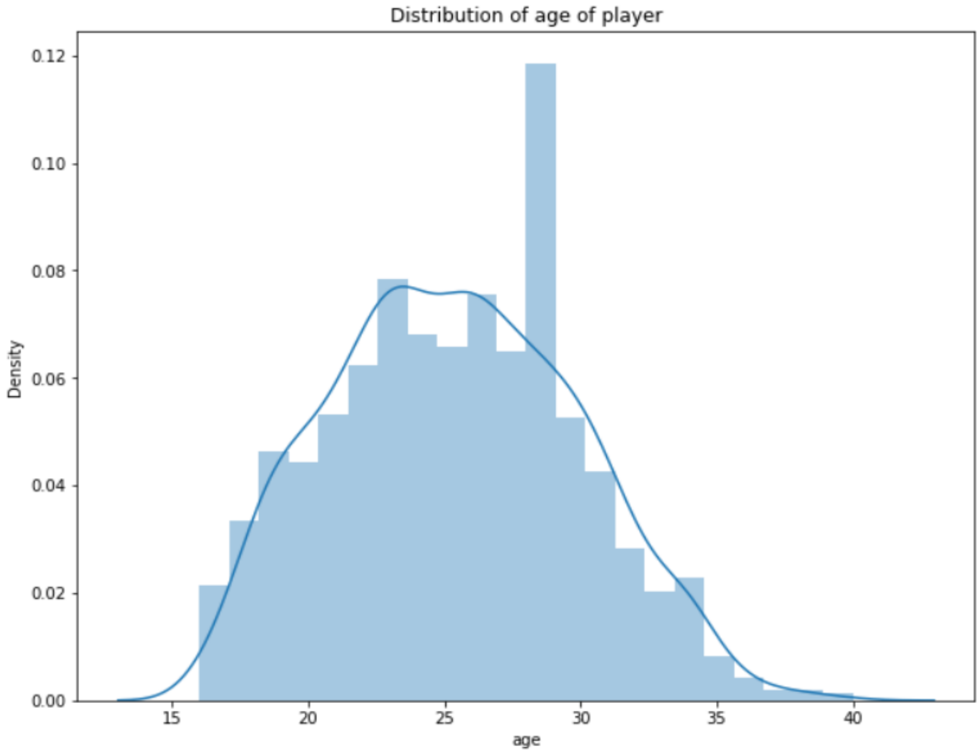


We can see that: The price of strikers is higher compared to other positions and the price of goalkeepers is always the lowest.

=> It means that the trend of football is to attack. Football teams and professionals focus on strikers. The proof is that most of the best players of the year play in the striker position.

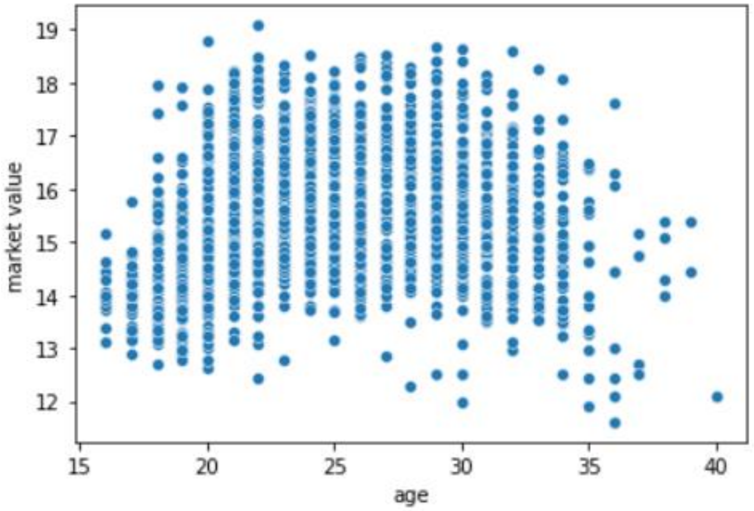
3. Age:

- See distribution of age:



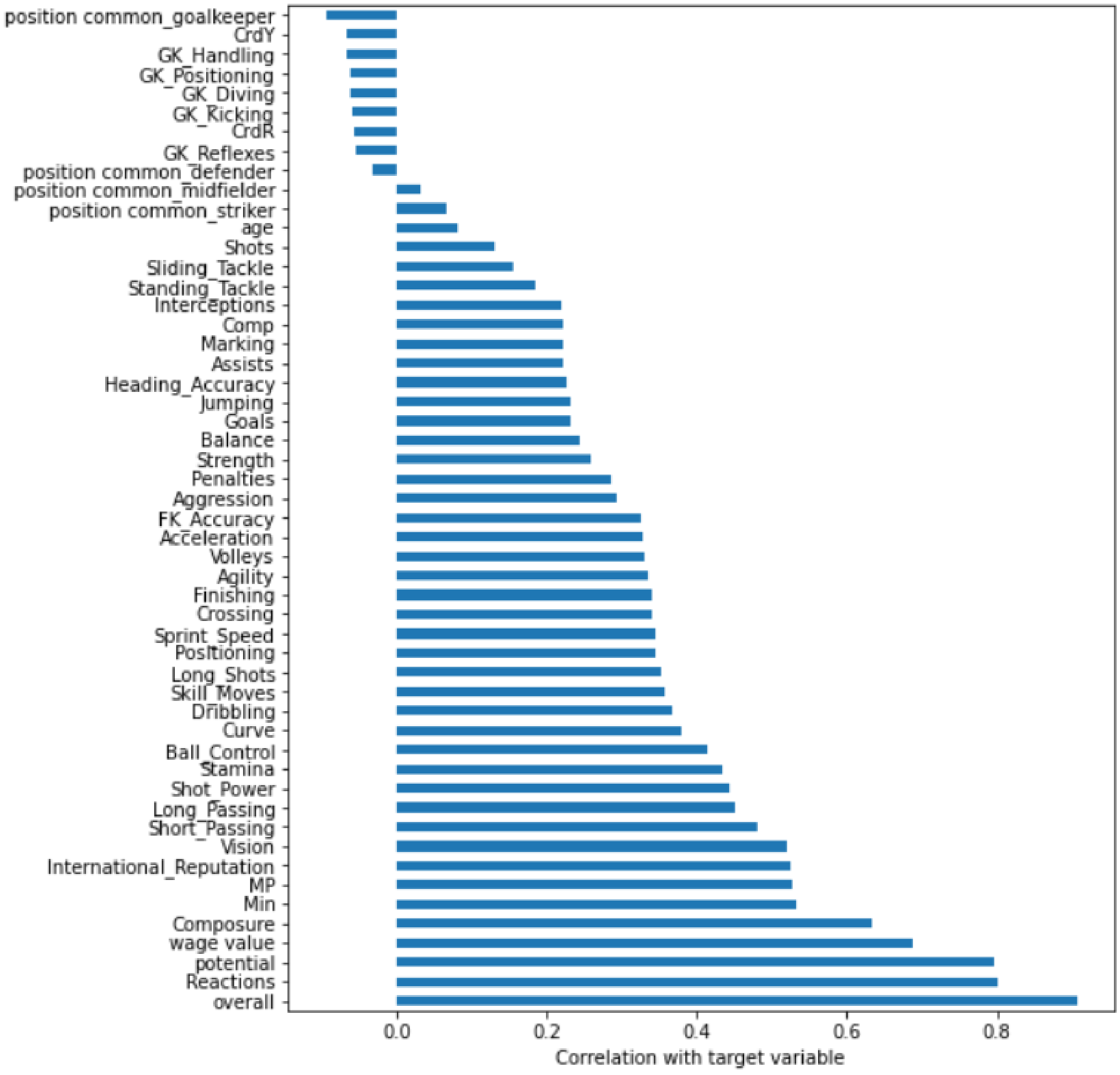
We see that: the distribution of age is near-normal distribution. Among them, there are about 88 players over 33 years old, notably Messi and Ronaldo.

- See correlation between age and market value:



⇨ we see that: Age doesn't have much of an impact on the prices of players in the top 5 leagues.

4. Correlation between independent variables and target variables:



We can see that: Some independent variables have high correlation with target variables, such as overall, reactions, potential, composure ,min, MP. Besides some independent variables have negative correlation with target variables, includes position common_goalkeeper, CrdY, GK_Handling, GK_Positioning, GK Diving, GK Kicking, CrdR, GK_Reflexes, position common defender.

V. Metric for evaluation models:

In this project, we use 2 metrics for evaluation models, namely root mean squared error and R2 score:

- Root mean squared error:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- R2 score:

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

where:

+ y_i : actual value
+ $\hat{y}_i$: predicted value
+ $\bar{y}$: mean of actual values

VI. Pre-processing and Split Train-Test

- First, Convert category columns into numeric columns:
  + Comp column (name of league): we use ordinal encoder, specially: Premier league into 1, La Liga into 2, Bundesliga into 3, Seria A into 4 and Ligue 1 into 5.
  + position common (it includes goalkeeper, defender, midfielder, striker): we use one-hot encoder
- Second, drop columns that haven't related in build model: name and preferred_foot columns.
- Third, feature scaling(just use for KNN - Lasso- Ridge): we use standardization for independent variables (x)
- Fourth, train- test split, 2/3 dataset used for training and 1/3 dataset used for testing.

VII. Modelling data

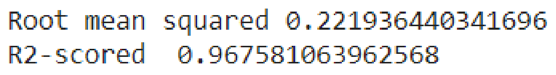
1. Lasso – Ridge – KNN:

Hyperparameter tuning for 3 models using GridSearchCV and optimizer: root mean squared error.

a. Lasso:

- Best parameter: alpha = 0.005.

⇨ Training model with hyperparameter, we have accuracy

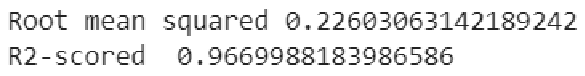


```
Root mean squared 0.221936440341696
R2-scored  0.967581063962568
```

b. Ridge:

- Best parameter: alpha = 0.4.

⇨ Training model with hyperparameter, we have accuracy

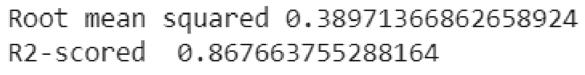


```
Root mean squared 0.22603063142189242
R2-scored  0.9669988183986586
```

c. K nearest-neighbor

- Best parameter: n_neighbors = 7, weights = 'distance', metric = 'mannhattan'

⇨ Training model with hyperparameter, we have accuracy



```
Root mean squared 0.38971366862658924
R2-scored  0.867663755288164
```

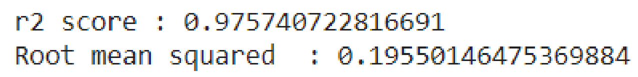
2. Decision Tree – Random Forest – Adaboost:

Hyperparameter tuning for 3 models using GridSearchCV and optimizer: root mean squared error

a. Decision Tree:

- Best parameter: criterion = 'friedman_mse', max-depth = 16, max_features = None, min_samples_leaf = 2

⇨ Training model with hyperparameter, we have accuracy

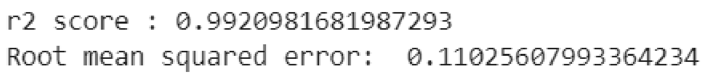


```
r2 score : 0.975740722816691
Root mean squared  : 0.19550146475369884
```

b. Random forest:

- Best parameter: n_estimators = 500, max_features = None, min_samples_leaf = 1

⇨ Training model with hyperparameter, we have accuracy

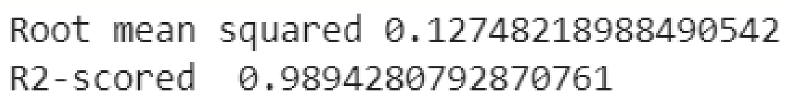


```
r2 score : 0.9920981681987293
Root mean squared error:  0.11025607993364234
```

c. Adaboost:

- Best parameter: base_estimator = DecisionTreeRegressor(max_depth=13,random_state=0), n_estimators = 94, learning_rate = 1.0, random_state = 3

⇨ Training model with hyperparameter, we have accuracy

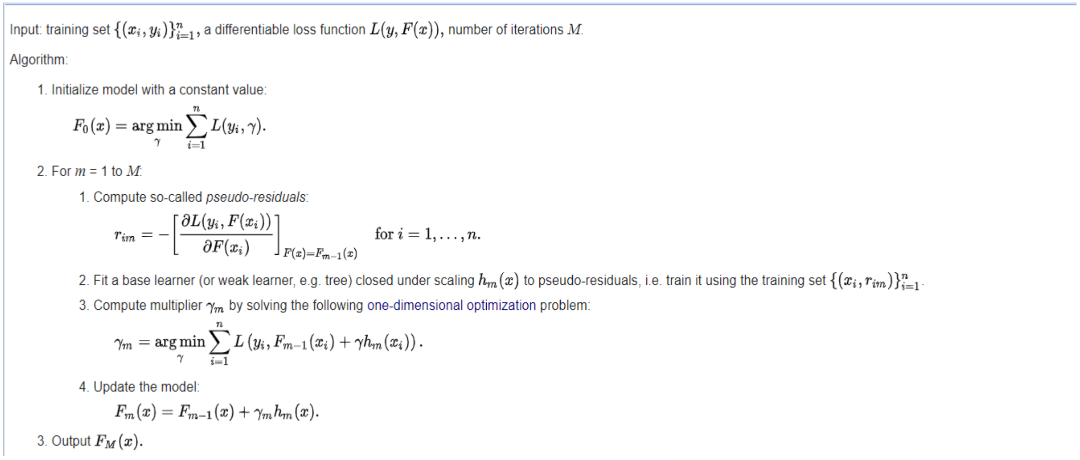


```
Root mean squared 0.12748218988490542
R2-scored  0.9894280792870761
```

3. Gradient Boosting: XGBoost and LightGBM

3.1. Theory of gradient boosting and its variations:

- A gradient-boosted model is built in stage-wise fashion as in other boosting methods. But it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.
- Generic gradient boosting method:



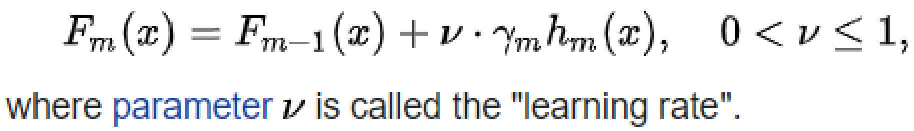
```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.

Algorithm:

  1. Initialize model with a constant value:
       $F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$
  2. For $m$ = 1 to $M$:
       1. Compute so-called pseudo-residuals:
            $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$
       2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
       3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
            $\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
       4. Update the model:
            $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
  3. Output $F_M(x)$.
```

- Regularization: some important parameters

+ Shrinkage:



$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1,$$

where parameter ν is called the "learning rate".

Empirically, it has been found that using small learning rate $v < 0.1$. But lower learning rate requires more iterations >=> increasing computation time.

+ number of trees: Increase M reduces the error on training set but setting it too high may lead to overfitting. An optimal value M is often selected by validation set.

+ Max depth: usually equal to 3 , we need to choose from 3 to 9 for max depth.

+ Penalize complexity of tree: L2 penalty on the leaf values can also added to avoid overfitting.

3.1.1. XGBoost:

- XGBoost is an open-source software library which provides a regulazing gradient boosing.
- Salient features of XGBoost which make it different from other gradient boosting algorithm include:

+ Clever penalization of trees

+ A proportional shrinking of leaf nodes

+ Newton Boosting

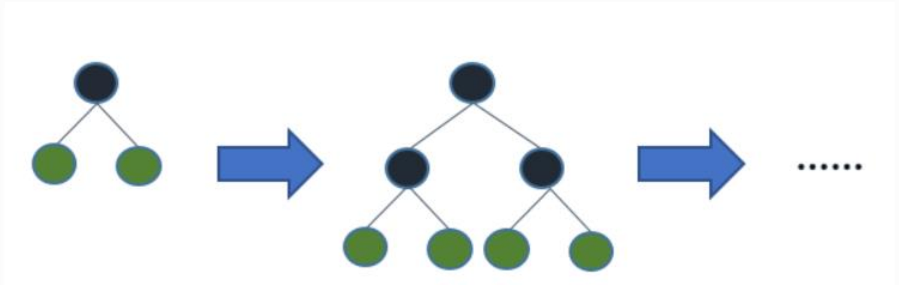
+ Extra randomization parameter

+ Implementation on single, distributed systems, and out-of-core computation

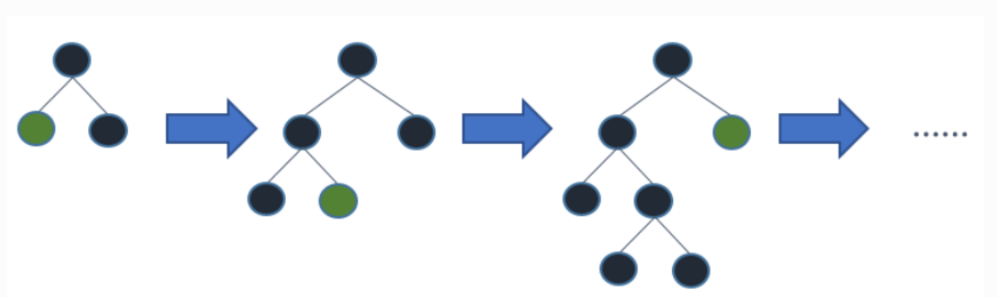
+ Automatic feature selection

3.1.2. LighGBM:

- LightGBM has many XGBoost's advantages, include sparse optimization, parallel training, multiple loss functions, regularization, bagging and early stopping.
- LightGBM doesn't grow a tree level-wise , instead of it grows trees leaf-wise. It chooses leaf that will yield the largest decrease in loss.



Level-wise tree growth



Leaf-wise tree growth

- LightGBM implements a highly optimized histogram-based decision tree learning algorithm.
- The LightGBM algorithm utilizes two novel techniques called Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) which allow the algorithm to run faster while maintaining a high level of accuracy

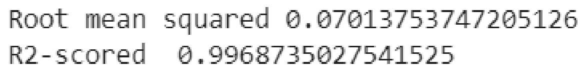
  + GOSS: that there is no native weight for data instance in GBDT, the instances with larger gradients will contribute more to the information gain.Thus, in order to retain the accuracy of the information, GOSS keeps the instances with large gradients and randomly drops the instances with small gradients.

  + EFB: is a near-lossless method to reduce the number of effective features. In a sparse feature space many features are nearly exclusive, implying they rarely take nonzero values simultaneously. EFB bundles these features, reducing dimensionality to improve efficiency while maintaining a high level of accuracy.

Hyperparameter tuning for 3 models using GridSearchCV and optimizer: root mean squared error.

a. XGBoost:

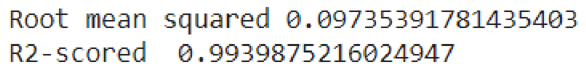
- Best parameter: n_estimators = 2000, learning_rate = 0.03, gamma = 0.0, max_depth = 3.
  (gamma : Gamma specifies the minimum loss reduction required to make a split. The values can vary depending on the loss function and should be tuned.)
  ⇨ Training model with hyperparameter, we have accuracy:



```
Root mean squared 0.07013753747205126
R2-scored  0.9968735027541525
```

b. LighGBM:

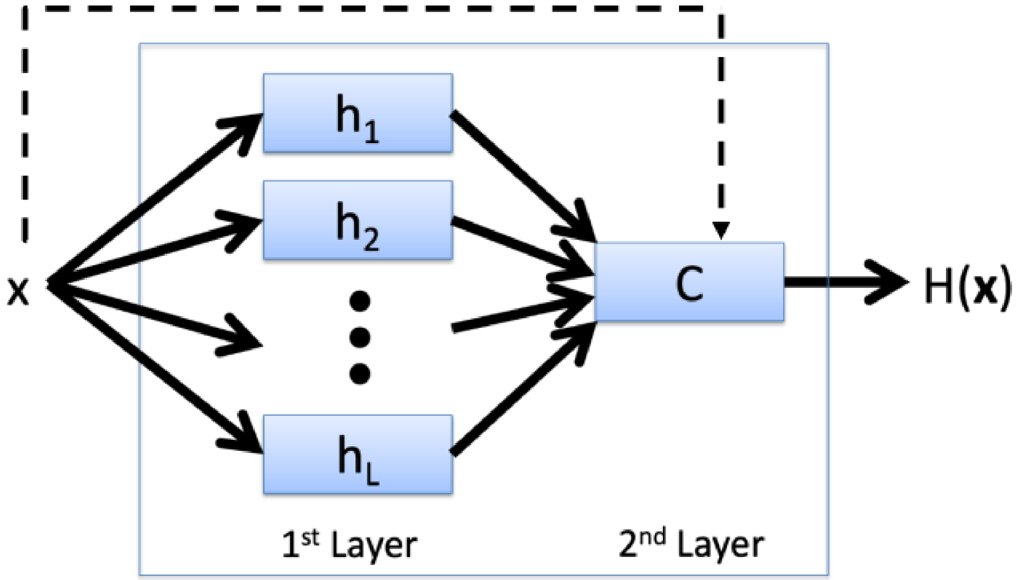
- Best parameter: n_estimators = 1000, learning_rate = 0.05, reg_alpha = 0.1, reg_lambda = 1, max_depth = 3
  (reg_alpha = L1 regularization, reg_lambda = L2 regularization)
  ⇨ Training model with hyperparameter, we have accuracy:



```
Root mean squared 0.09735391781435403
R2-scored  0.9939875216024947
```

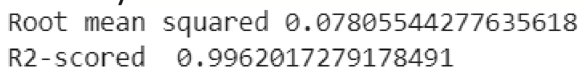
4. Stacking:

Hyperparameter tuning with GridSearchCV, optimizer : root mean squared error:



Here is an image of stacking with 2 layers, so in this project, we also use stacking with 2 layers.

- Layer 1: we use 4 models, namely: Lasso, Ridge, Decision Tree and Random Forest. After we predict for 4 models, the result of 4 models will add into independent variables(x) as 4 new features.
- Layer 2: we use XGBoost for hyperparameter turning
  - ⇨ Best parameter: n_estimators = 6000, learning_rate = 0.01, gamma = 0, max_depth = 3
  - ⇨ Training models with hyperparameter, we have accuracy:

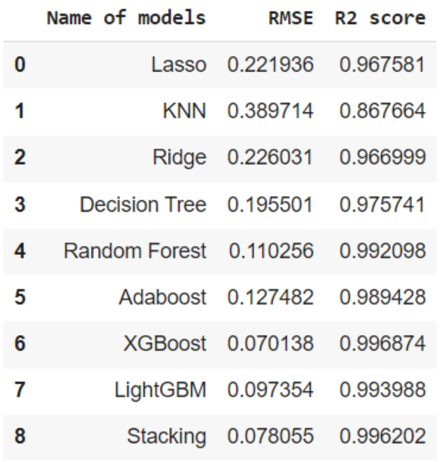


```
Root mean squared 0.07805544277635618
R2-scored  0.9962017279178491
```

VIII. Summary result and analysis with best models:

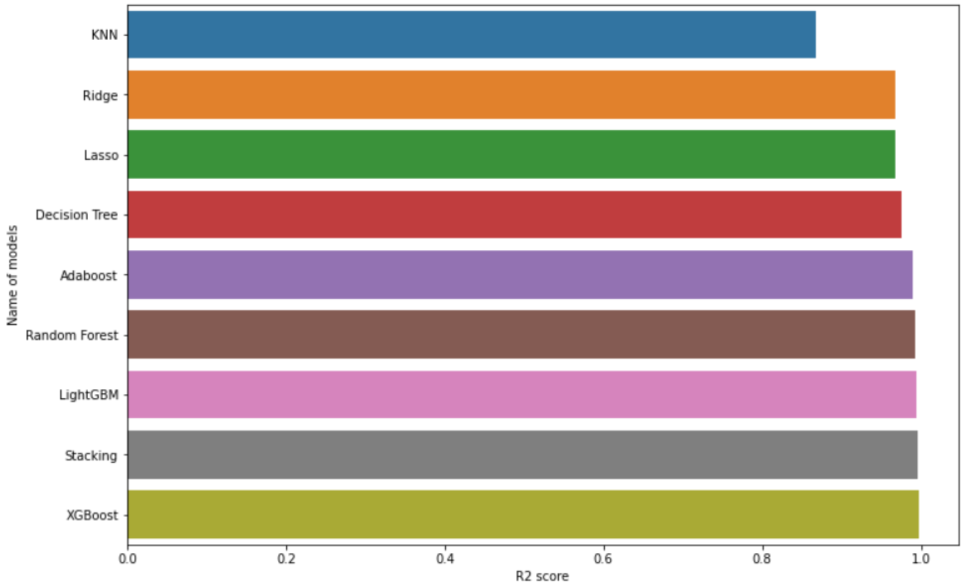
1.Summary result:

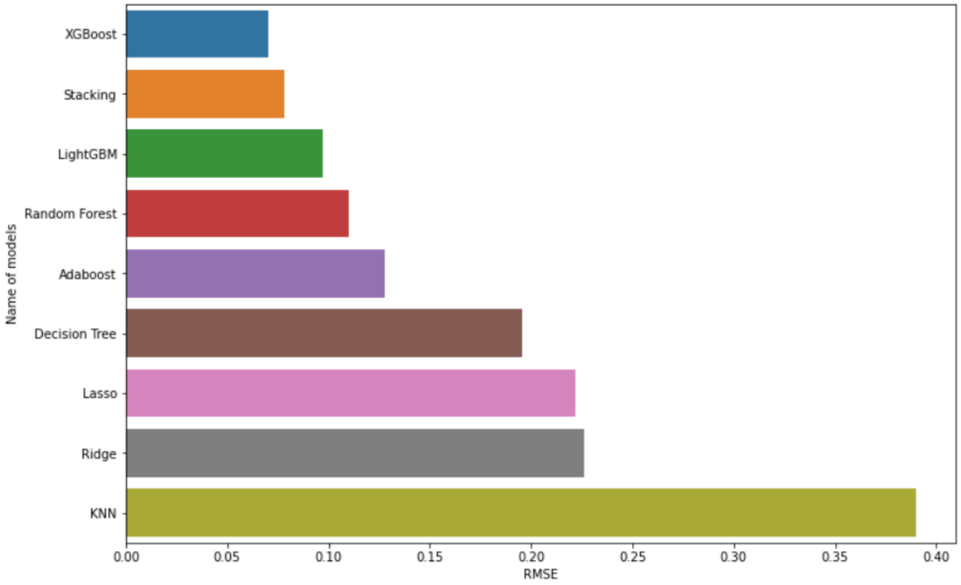
- Tables perform the accuracy of 9 models



	Name of models	RMSE	R2 score
0	Lasso	0.221936	0.967581
1	KNN	0.389714	0.867664
2	Ridge	0.226031	0.966999
3	Decision Tree	0.195501	0.975741
4	Random Forest	0.110256	0.992098
5	Adaboost	0.127482	0.989428
6	XGBoost	0.070138	0.996874
7	LightGBM	0.097354	0.993988
8	Stacking	0.078055	0.996202

- Barplot for RMSE and R2 -score:



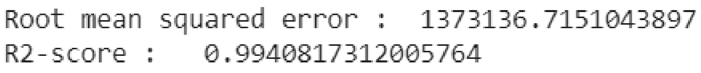


- ⇨ we can see that: XGBoost has has the smallest RMSE and the largest R2_score
- ⇨ XGBoost is best model, so we decided to apply this model for our problem, we will use it for analysis result.

2. Analysis with best models:

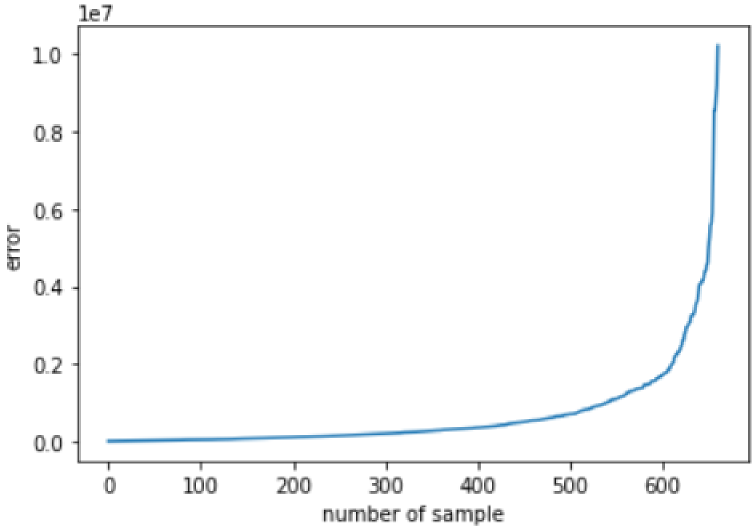
- Because we used to log transformation technique for market value variables, so we need to evaluate actual values with best model.

- mean squared error and R2 score for actual values and predicted values:



```
Root mean squared error :  1373136.7151043897
R2-score :   0.9940817312005764
```

- plot the error between actual values and predicted values:

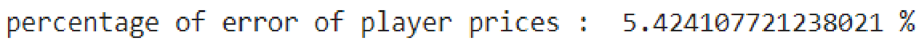


⇨ we can see that : we have approximately 610 player prices that the error of its prediction < 2 million euro.

- Finally, I use a new metric to calculate the percentage of error of player prices:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i - y_i|}{y_i}$$

⇨ Apply new metric for model, we have :



```
percentage of error of player prices :  5.424107721238021 %
```

⇨ It means that: If a player is really worth 100 million euros, our best model would predict that player's value between 100 - 5.4 million euros to 100 + 5.4 million euros.

⇨ Here is good prediction.

IX. Future:

In the future, we will search and scrape data for all players in the world, not just the top 5 European leagues. Then will predict the price for all those players

X. References:

https://docs.scrapy.org/en/latest/intro/tutorial.html

https://www.codementor.io/@mgalarny/using-scrapy-to-build-your-own-dataset-cz24hsbp5

https://en.wikipedia.org/wiki/Gradient_boosting

https://en.wikipedia.org/wiki/XGBoost

https://en.wikipedia.org/wiki/LightGBM

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

https://xgboost.readthedocs.io/en/stable/

https://lightgbm.readthedocs.io/en/v3.3.2/

https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

https://docs.scrapy.org/en/latest/intro/tutorial.html

https://www.codementor.io/@mgalarny/using-scrapy-to-build-your-own-dataset-cz24hsbp5

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/