Machine Learning 2022
Report - Player Value Prediction

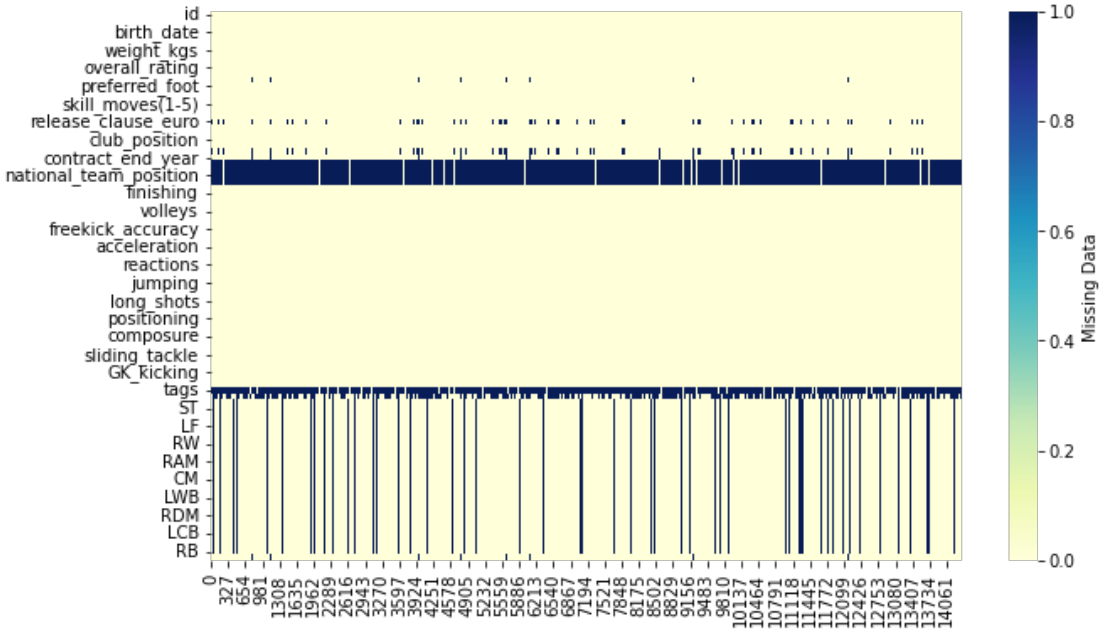
Milestone 1. *Team: CS_37*

Before applying a model to our dataset, it's important to pre-process the data in order to get a model as accurate as possible. The preprocessing techniques are: 1- Data cleaning (handling null values), 2-Data transformation, 3- Outlier detection, 4- Feature selection.

1- **Data cleaning:**

We clean the data by dropping columns or rows that contain too many null values or imputing by mean/median for numerical values and imputing by the mode for categorical values (most repeated/common category).

To deal with the null values in our dataset, we first checked to see the columns that contained null values:



The following heatmap helps us to visualize which features have null values and how many null values are in each, we can see that the features: 'tags' and 'national_team_position' include more null values than not, so those feature should be dropped. The heatmap however doesn't show us all the features, and it would be a tedious process to try to figure out whether to drop just by looking at the graph. Instead, we checked all features that

contain nulls and dropped all that included over 20% null values.

```
In [64]: columnsIter = pd.DataFrame(dataframe, columns=dataframe.columns, index=[dataframe.index])

         for (columnName, columnData) in columnsIter.iteritems():



             if(dataframe[columnName].isnull().sum()!=0):

                 if(dataframe[columnName].isnull().sum() > 0.2 * 14363):
                     print(columnName," ", dataframe[columnName].isnull().sum())
                     dataframe = dataframe.drop([columnName], axis=1)

         national_team   13675
         national_rating   13675
         national_team_position   13675
         national_jersey_number   13675
         tags   13242
         traits   7859
```

These were the features to contain too many null values. Now our dataset includes the following columns:

```
In [64]: columnsIter = pd.DataFrame(dataframe, columns=dataframe.columns, index=[dataframe.index])

         for (columnName, columnData) in columnsIter.iteritems():



             if(dataframe[columnName].isnull().sum()!=0):

                 if(dataframe[columnName].isnull().sum() > 0.2 * 14363):
                     print(columnName," ", dataframe[columnName].isnull().sum())
                     dataframe = dataframe.drop([columnName], axis=1)

         national_team   13675
         national_rating   13675
         national_team_position   13675
         national_jersey_number   13675
         tags   13242
         traits   7859
```

And the columns that contain null values are:

Feature	null values	data type
wage	191	float64
release_clause_euro	1482	float64
club_team	12	object
club_rating	12	float64
club_position	12	object
club_jersey_number	12	float64
club_join_date	1565	object
contract_end_year	287	object
LS	1672	object
ST	1672	object
RS	1672	object
LW	1672	object
LF	1672	object
CF	1672	object
RF	1672	object
RW	1672	object
LAM	1672	object
CAM	1672	object
RAM	1672	object
LM	1672	object
LCM	1672	object
CM	1672	object
RCM	1672	object
RM	1672	object
LWB	1672	object
LDM	1672	object
CDM	1672	object
RDM	1672	object
RWB	1672	object
LB	1672	object
LCB	1672	object
CB	1672	object
RCB	1672	object
RB	1672	object
value	198	float64

This brings us to the 2nd step of data cleaning, imputing. This will

require us to define whether the data is numerical or categorical, as the table shows, columns columnsIter2 = pd.DataFrame(dataframe, columns=['LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM', 'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'contract_end_year', 'club_join_date', 'club_position']

To deal with the categorical values, we must inspect the data type to see the easiest way to convert it to numerical, for the columns: ['LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM', 'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB'], the data is written in the form of re: [0-9]* /+ [0-9]* , this means that a value of 62+2 means that the value can range anywhere from 62 to 64, it's redundant as the highest value is 3 which means the range is limited and doesn't affect the values by a significant amount. We will handle this data by **transforming the data**(2nd preprocessing step) and removing the +. We will replace the + with . (ie. 62+2 will become 62.2), which is a float64 value that can be applied to our model. We will do this by iterating over our values and using **.str.replace('+', '.')**. This will however transform our data to a String datatype so we will parse it to **pd.to_numeric()** function in order to change it to a numerical value. The result will be:

After we have transformed the data to numeric, we can now use **imputing by the mean** in order to replace the null values and complete our data, we will also apply this imputing method to all our numerical data.

The result will be:

Feature	null values	data type
LS	0	float64
ST	0	float64
RS	0	float64
LW	0	float64
LF	0	float64
CF	0	float64
RF	0	float64
RW	0	float64
LAM	0	float64
CAM	0	float64
RAM	0	float64
LM	0	float64
LCM	0	float64
CM	0	float64
RCM	0	float64
RM	0	float64
LWB	0	float64
LDM	0	float64
CDM	0	float64

```
RDM                  0          float64
RWB                  0          float64
LB                   0          float64
LCB                  0          float64
CB                   0          float64
RCB                  0          float64
RB                   0          float64
wage                 0          float64
release_clause_euro  0          float64
club_rating          0          float64
club_jersey_number   0          float64
value                0          float64
```

2- **Data Transformation:**

What we have left now are the two feature 'club_join_date' and 'contract_end_year', by inspecting the values, we find that the values are in date format. Most of the values of contract_end_year have values of only the year, with a few containing values written in dd-mm-yy format, we can transform the data to get only the year of those few values so all values of contract_end_year are years only by applying the following code:

```
In [249]: strng=map(str, lst)
          for month in (strng):

              if(month!=None):
               month=re.sub('[0-9]*-[A-Z][a-z]*-', '20', month)

              mylist.append(month)
```

This code will loop over our values and check to see if the following regular expression format is found, if found it will replace it with '20' (ie. 30-Jun-19 will be transformed to 2019). Then value will be parsed to to **pd.to_numeric()** function in order to change it to a numerical value. Then we can drop the original column and replace it with a new one made from the mylist list.

The last feature to deal with now is 'club_join_date', the values of this column are written in the date format mm/dd/yy, to transform this data we can create 3 individual features: months, days and years.

```
In [246]: for month in (strng):

              if(month!= 'nan'):
               ordered =month.split("/", 2)
               month = ordered[0]
               day=ordered[1]
               year=ordered[2]
               months.append(month)
               days.append(day)
               years.append(year)
```

This loop will iterate over our values and use the '/' character to split our date string into 3 different strings, (ie. 12/30/2020 will be ordered=["12","30","2020"]), then the values of each element of the array will be appended to a list which we will create a new feature from. After dropping the original column

and parsing the new columns to **pd.to_numeric()**, we now have 3 new columns in our dataset of numerical values int64:

```
In [273]: dataframe.head()
```

Out[273]:

weak_foot(1-5)	...	position_first	position_second	position_third	upper_work_rate	down_work_rate	months	days	years	contract_end_year	num_of_work_rate
3	...	7	11	0	1	3	7	1	2018	2020	0.333333
3	...	1	0	0	2	3	1	31	2019	2019	0.666667
3	...	14	7	0	1	3	7	24	2017	2020	0.333333
3	...	6	9	0	1	1	7	1	2012	2020	1.000000
3	...	13	11	0	3	3	2	11	2018	2021	1.000000

Before moving on to the next step of pre-processing, we still have 4 features that we cannot apply to our model, those features are:['id', 'name', 'full_name']. Their values are categorical and unique for each column. These columns will be dropped from the dataset entirely.

As for the last feature ‘birth_date’, we applied the same pre-processing techniques applied to ‘club_join_date’ but only kept the months column as the year will be similar to ‘age’.

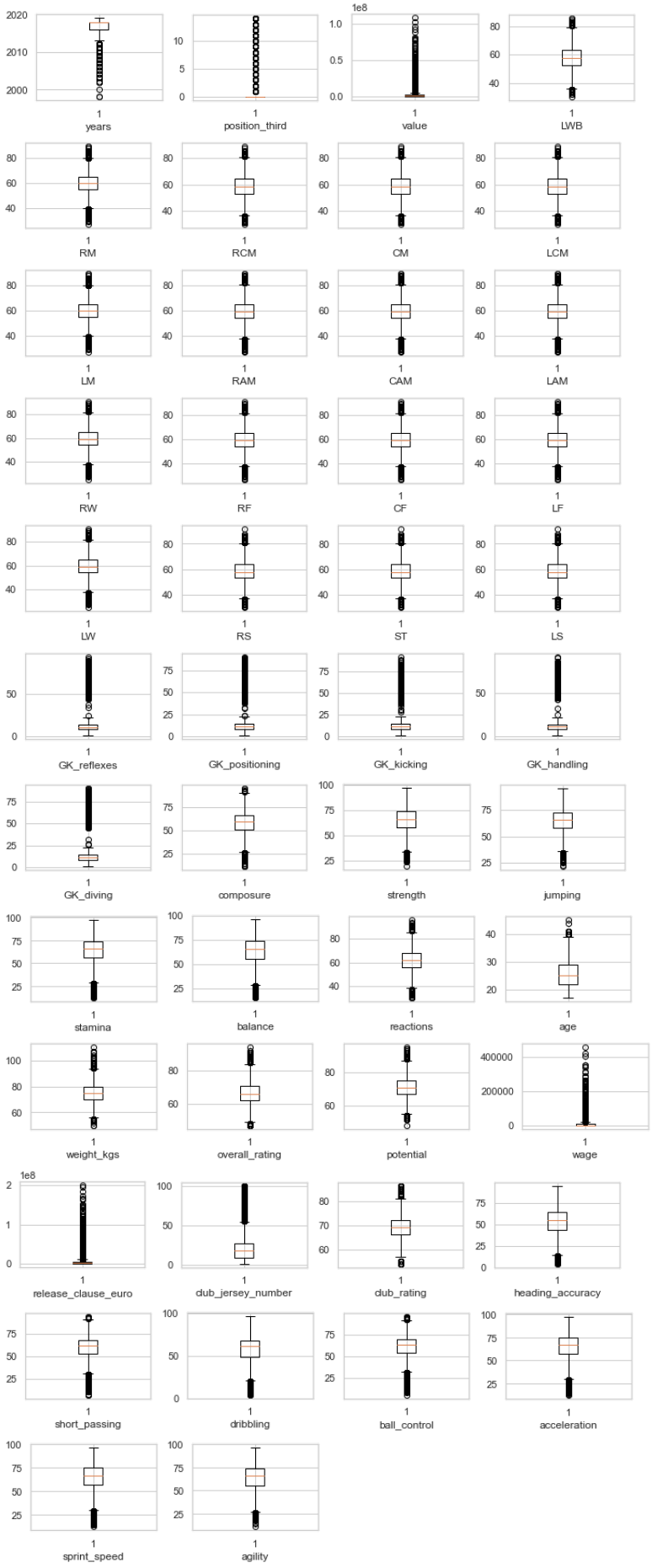
Before applying outlier detection, we can analyze the data and select an initial value of features. By applying correlation and selecting all feature which have a correlation with the ‘value’ **feature of more than 50%, our initial features are:** ['overall_rating', 'potential', 'wage', 'international_reputation(1-5)', 'release_clause_euro', 'club_rating', 'reactions']

The size of the data set that we used for the model is: 14363 rows × 7 columns
Training size = 80%
Test size = 20%
Polynomial degree =3
This gave us the following results:
Score = 0.9560107473923647
MSE = 1906598410581.646
Training time: 0.04074907302856445 Seconds
We will use this as a guide to check with our outliers.

3- **Outlier detection:**

The next step of pre-processing is outlier detection. We will use box-plots to represent the outliers found in our data as they’re simple to understand. We now have 89 features of all numerical data, a lot of which we had to handle ourselves, so the possibility of outliers is large.

After plotting each feature, this was the result:

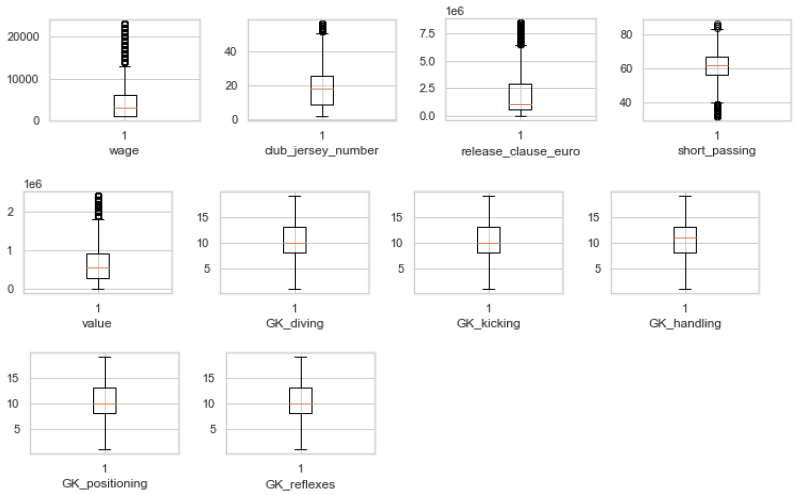


These are not all the box plots but enough to work with for now, now it's time to remove the outliers, we can do that by

dropping a row containing the outlier or by entirely removing the column. We will divide our outlier removal into iterations to test for scores after each iteration.

First iteration:
Consists of the following features as they are they appear to contain the biggest amount of outliers:
wage, release_clause_euro, club_jersey_number,short_passing,GK_diving, GK_handling,GK_kicking,GK_positioning,GK_reflexes and value, we can drop the features entirely however this will not work on the value feature as that is what we are trying to predict with our model. We will instead test once by dropping the rows and by

First test:
After dropping the rows of those features by using IQR analysis, we are now left with a dataset of 9043 rows × 89 columns, meaning 5320 rows were removed in total. If we check the outliers of the first iteration features now, we can see that they have decreased drastically:



However, when applying the new dataset after removing the outliers specific to these features, our model score decreases.
After removing outliers:
The features which have a correlation with the 'value' feature of more than 50%, our initial features are:

['overall_rating', 'potential', 'wage', 'release_clause_euro', 'ball_control', 'reactions', 'composure', 'LCM', 'CM', 'RCM']

The size of the data set that we used for the model is:
9043 rows × 10 columns

Polynomial degree =3
Training size = 80%
Test size = 20%
This gave us the following results:
Score = 0.897477905661259
MSE = 27742395663.53798
Training time: 0.08725333213806152 Seconds

Second test:
If we remove the features of the first iteration (excluding 'value' and 'wage'), and handle 'value' 'wage'outliers separately by dropping the rows, we get the following score:
The features which have a correlation with the 'value' feature of more than 50%, our initial features are:
['overall_rating', 'wage', 'reactions']
The size of the data set that we used for the model is:
11424 rows × 3 columns
Training size = 80%
Test size = 20%
Polynomial degree =3
This gave us the following results:
Training time: 0.004843950271606445 Seconds
Score: 0.6655598063937211
MSE = 145349004067.8879

Conclusion: First test results have a higher score so we will continue on that.

2nd iteration features:
Removing outliers for the following features:
['ball_control', 'dribbling', 'club_rating', 'stamina','RAM', 'CAM', 'acceleration', 'sprint_speed', 'agility', 'weight_kgs', 'age']

First test:
The result is a dataset of **7355 rows × 89 columns**, which previously was 9043 rows. Let's evaluate this test:

The features which have a correlation with the 'value' feature of more than 50%, our initial features are:
['overall_rating', 'potential', 'wage', 'release_clause_euro', 'ball_control', 'reactions', 'composure', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM']
The size of the data set that we used for the model is:
7355 rows × 22 columns
Polynomial degree =3
Training size = 80%
Test size = 20%
This gave us the following results:
MSE = 24120872986.52409
Training time: 3.8800628185272217 Seconds
Score: 0.9084049990612153
Poly fitting time: 2mins 51.1 secs

2nd test:
We'll add a few more features to have the outliers removed:
['LF','RM', 'RW','LAM', 'LM']

Let's evaluate this test:
The features which have a correlation with the 'value' feature of more than 50%, our initial features are:
['overall_rating', 'potential', 'wage', 'release_clause_euro', 'ball_control', 'reactions', 'composure', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM']
The size of the data set that we used for the model is: 7283 rows × 23 columns

With polynomial regression model:
Polynomial degree =3
Training size = 80%
Test size = 20%
This gave us the following results:
Poly fit time: 168.34758305549622 Seconds
Training time: 3.9204537868499756 Seconds
Score: 0.8816406689473453
MSE: 32736138180.780083
With linear regression model:
Training time: 0.0038661956787109375 Seconds

Score: 0.7543599275586523
MSE: 64687514897.436226

The first test has given us better results, so let's attempt a third where we only remove outliers for: ['ball_control', 'dribbling', 'club_rating', 'stamina', 'acceleration', 'sprint_speed', 'agility', 'weight_kgs', 'age'],

And we will also drop the following columns from the dataset completely:
['RM','LAM', 'LM','RAM']

Let's evaluate this test:
The features which have a correlation with the 'value' feature of more than 50%, our initial features are:
['overall_rating', 'potential', 'wage', 'release_clause_euro',
'ball_control', 'reactions', 'composure', 'LS', 'ST', 'RS', 'LW', 'LF',
'CF', 'RF', 'RW', 'CAM', 'LCM', 'CM', 'RCM']
The size of the data set that we used for the model is: 7365 rows × 19 columns

With polynomial regression model:
Polynomial degree =3
Training size = 80%
Test size = 20%
This gave us the following results:
Poly fit time: 33.35180306434631 Seconds
Training time: 1.117898941040039 Seconds
Score: 0.9077751442164206
MSE: 27052895954.676308

With linear regression model:
Training time: 0.0052149295806884766 Seconds
Score: 0.7613856447194587
MSE: 69994246907.19872

The 3rd iteration will attempt to remove outliers for the following features:

'ball_control', 'stamina', 'acceleration', 'sprint_speed', 'agility', 'weight_kgs', 'age', 'overall_rating', 'reactions', 'composure' , 'CF', 'RF', 'LW', 'RS', 'position_third','years','CM',

'LCM','wage','value','positioning','vision','shot_power','long_passing','short_passing','release_clause_euro','potential'

It will also drop these columns from data set:
'weak_foot(1-5)', 'position_third','years','CM', 'LCM', 'jumping', 'balance'

Let's evaluate this test:
The features which have a correlation with the 'value' feature of more than 50%, our initial features are:
['overall_rating', 'potential', 'wage', 'release_clause_euro',
'ball_control', 'reactions', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF',
'RW', 'LAM', 'CAM', 'RAM', 'LM', 'RCM', 'RM']
The size of the data set that we used for the model is: 5561 rows × 20 columns

With polynomial regression model:
Polynomial degree =3
Training size = 80%
Test size = 20%
This gave us the following results:
Poly fit time: 54.98618292808533 Seconds
Training time: 1.518401861190796 Seconds
Score: 0.9475632982264189
MSE: 6955957813.451437

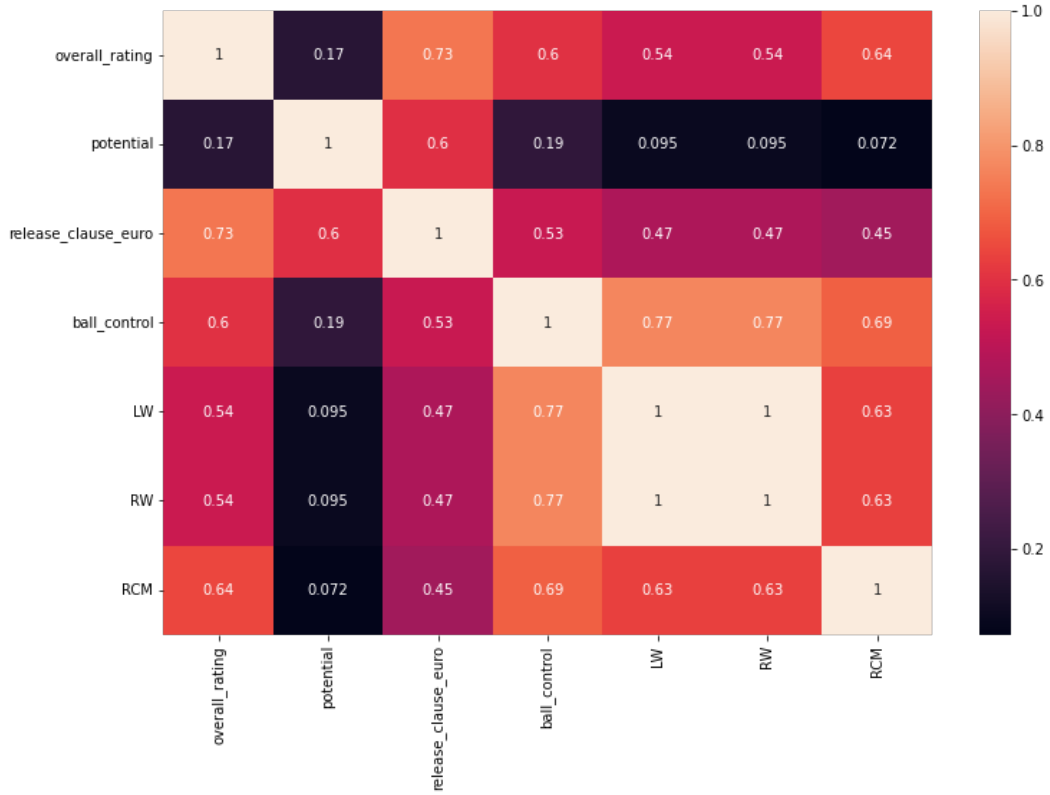
With linear regression model:
Training time: 0.0066187381744384766 Seconds
Score: 0.8048697071738202
MSE: 25884886713.24452

As we can see, to find the best model after IQR analysis it take multiple tries to get right, we printed the box plots for the remaining features after every iteration and applied the

results to our model to test for the score. Our most successful attempt was the following:

Using outlier detection we dropped the features: 'dribbling', 'club_rating', 'jumping','balance, 'years','CM', 'LCM', 'positioning','long_passing','short_passing', 'aggression','long_shots','LWB','club_jersey_number','num_of_work_rate','penalties','club_position', 'skill_moves(1-5)', 'down_work_rate', 'RM','LM','RAM','CAM','LAM' 'acceleration', 'strength', 'stamina', 'weak_foot(1-5)','preferred_foot', 'international_reputation(1-5)','position_third', 'volleys','sprint_speed', 'heading_accuracy', 'LF','CF','RF'] our dataset now contains 52 features out of originally 92 and 4324 rows out of originally 14343.
Dimensions(4324 rows x 52 columns)
The features this data has high correlation with (over 50%) are: **['overall_rating', 'potential', 'release_clause_euro', 'ball_control', 'LW', 'RW', 'RCM']**

Heatmap to visualize the correlation:



The results of this new dataset once we applied it to our models:
Training size = 80%
Test size = 20%
For our linear regression model the result was:
1- **Linear Regression:**
Training time: 0.009855985641479492 Seconds
Score: 0.8407758216481678
MSE: 14748448610.203592
root of MSE 121443.19087624301

Cross validation score is 0.9594998305574858

Compared with applying no outlier detection/extraction where our results were:

Training time: 0.013540029525756836 Seconds
Score: 0.7015760206403678
MSE: 12934402177766.615
root of MSE 3596442.989644993
Cross validation score is 0.9622418948371704

2- **Polynomial Regression (of degree 4):**

Poly fit time: 29.5032377243042 Seconds
Training time: 0.06858515739440918 Seconds
Score: 0.972199954608113
MSE: 2575033170.638141
Root of MSE 50744.78466441789
Cross validation score is 0.9594998305574858

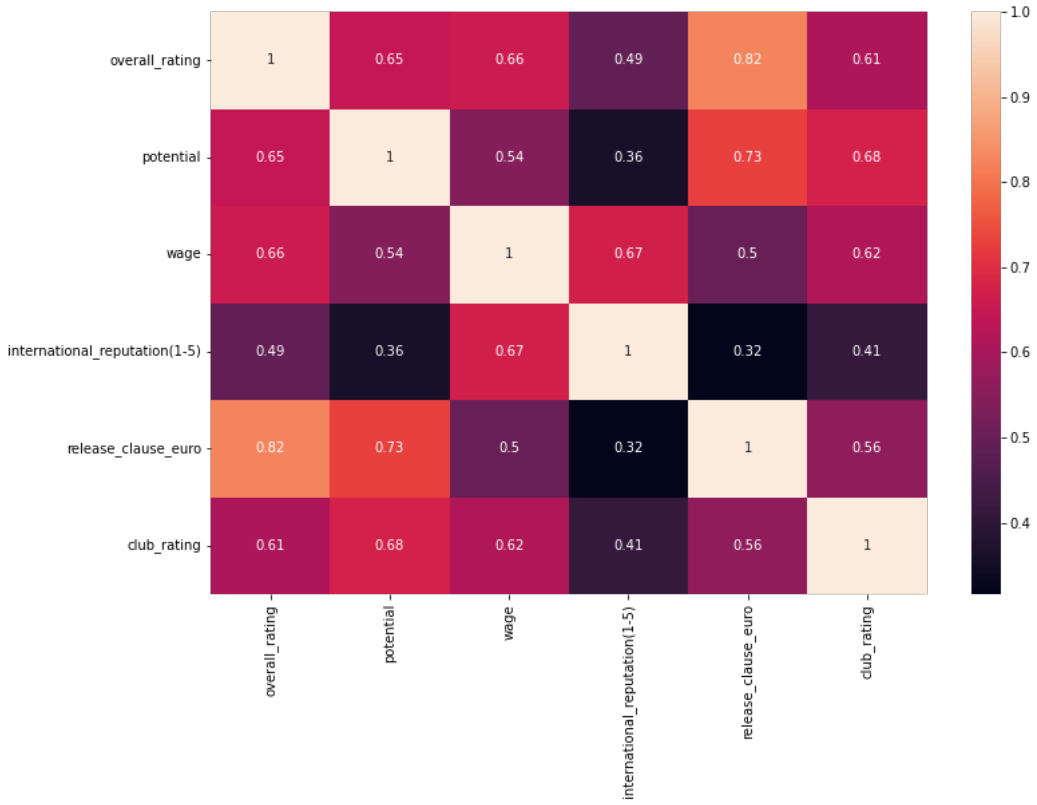
Compared with dataset results before applying outlier detection/extraction:

Poly fit time: 33.560181856155396 Seconds
Training time: 0.3470292091369629 Seconds
Score: 0.9502607920270738
MSE: 2155815096715.4768
Root of MSE 1468269.422386599
Cross validation score is 0.9622418948371704

The initial result was applied on the original dataset after all features were turned to numerical values we can use.
The following features had the highest correlation with the value at first:
'overall_rating', 'potential', 'wage', 'international_reputation(1-5)', 'release_clause_euro', 'club_rating', 'reactions'

And the dimension were 14363 rows × 7 columns

Heatmap plotting the correlation of features:



The two regression models we used were linear regression and polynomial regression.

Linear regression:

Linear regression is a linear model which tries to calculate how can we can obtain the input and the output through a linear relationship (ie. Y = mx +b). It's in the form of a straight line. The good thing about linear regression is how simple but efficient it is when applied to model with datapoints that form anything resembling a straight line. However, its simplicity can't make up for the limited models it efficiently works on. Datapoints are usually scattered in a way that can't possible

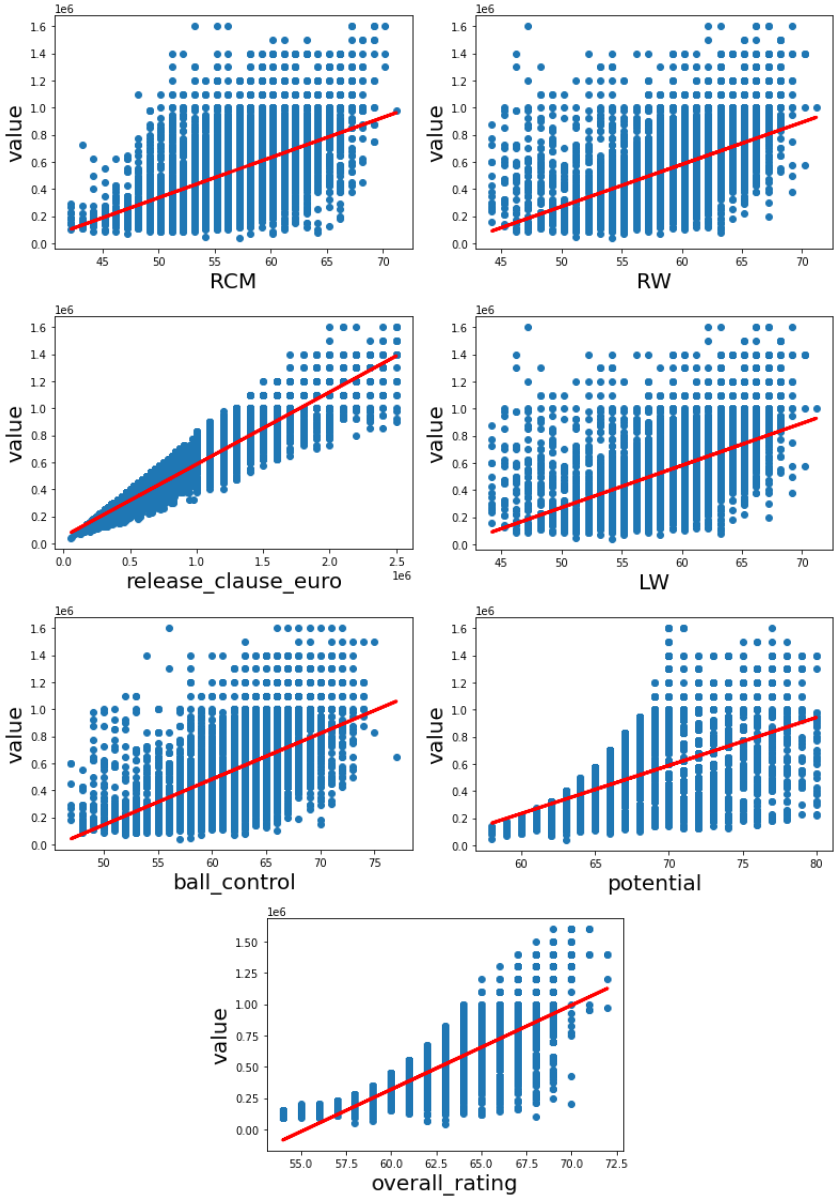
resemble a straight line. This is where a second model comes in.

Polynomial regression:

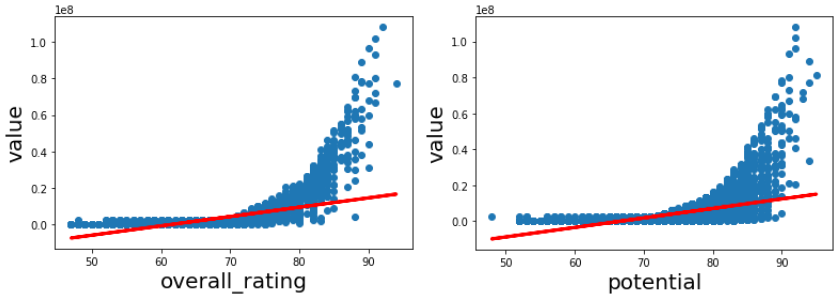
Though it's also a regression model like linear regression, polynomial regression is a lot more efficient in the real world which doesn't consist of only straight lines. Polynomial regression has the same goal, to find a way to predict the output through an input, or map the independent value to the dependent, but polynomial regression doesn't look for just a straight line to find the relationship. Instead, polynomial regression can increase its power to however large in order to map out the relationship (ie.). This comes with consequences though, as polynomial regression can encounter a problem of over-fitting, where it learns its data so well that it can't predict the values as it only knows the values it was given.

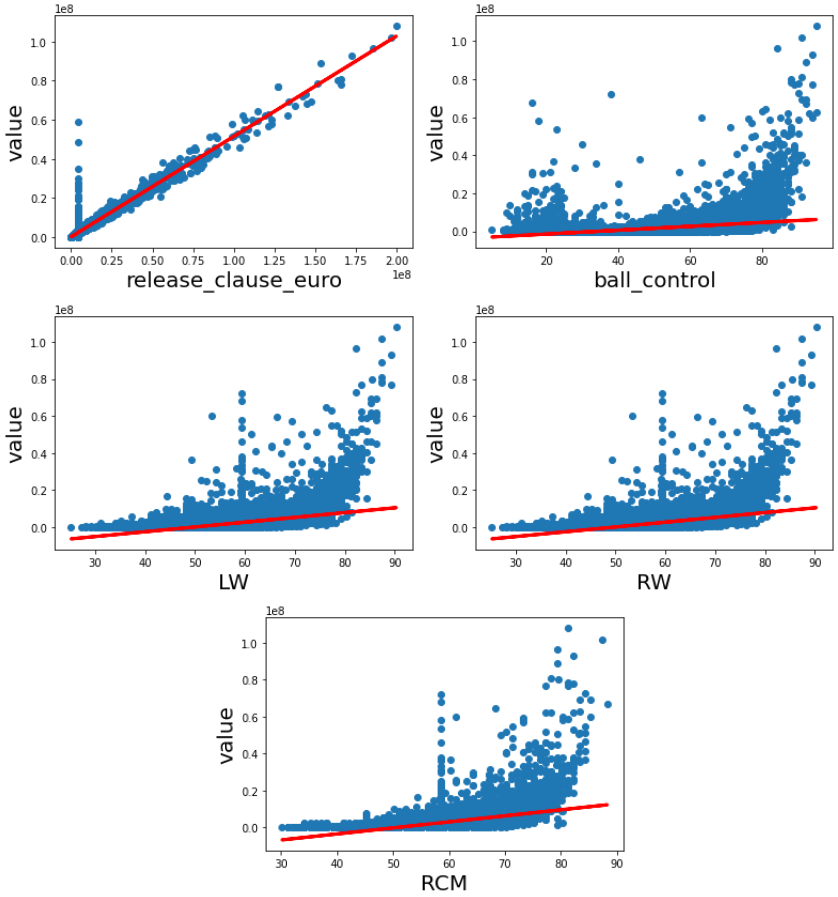
Through our experiments, you can notice the accuracy and the results become better as we use polynomial regression compared with linear regression.

Graphing linear reg: For the final dataset



Graphing linear reg: For the pre-processed dataset before removing outliers:





Conclusion: The technique used to improve the result was outlier detection and removal. Initially we had assumed that there was no use of using outlier detection and we assumed it would be okay to leave it out. We also initially assumed pre-processing didn't affect results as much and it was the model which affected the result, but we have been working on the same model since the start and it only improves when we apply more preprocessing techniques. We initially assumed features like 'reactions' and 'wage' would be correlated highly but as we removed outliers it became more clear that they don't have a high correlation with the player value.