Machine Learning Using Python & Spark

ADARSH RAJ, A21001

Contents

PROBLEM STATEMENT	DATA ANALYSIS	MODELLING	RESULTS
Why we need to solve it?	What does the data look like?	Using Linear Regression ML model	Comparison
What will we gain by solving it?	What are we going to do with the data?	Using all columns	Did the end justify the means?
Okay great, how to solve it?	Analysis & Exploration	Using all non-highly correlated columns	

Problem Statement

--Why we need to solve it?

- So, FIFA belongs to a very popular e-Sports game segment and there are lots of virtual tournaments that are held throughout the world which involves a large prize pool

- e-Sports player face a lot of difficulty while choosing the kind of players they would want for their Ultimate team event online as that involves lot of research and selecting player which suits their respective needs.

Problem Statement

--What will we gain by solving it?

- This will help e-Sports players save a lot of time while selecting the kind of players they would want for their team under limited budget but also tick marks most of their requirements

- A sense of personal satisfaction. I have been a huge fan of FIFA game and actively play game in career mode and Ultimate team mode wherein earlier I used to waste a lot of time selecting the kind of players I would need for myself. Say, no more, I got the model for myself now!

Problem Statement

--Okay great, how to solve it?

We will be using Spark & Python in Google Colab environment to solve our problem at stake here.

Approach we will be following –

- Initiating Spark environment and loading required libraries & modules when needed
- Loading & Understanding the data
- Exploratory Data analysis
- Using ML model to get the desired results

Data

--What does the data look like?

The original dataset structure is as below. Note that there are 19239 records and 100+ columns.

```python
# Reading data from CSV file
players = spark.read.csv('players_22.csv', sep=',', header=True, inferSchema=True, nullValue='NA')

# Get number of records
print("The data contain %d records." % players.count())

# View the first five records
players.show(5)
```

The data contain 19239 records.

sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value_eur	wage_eur	age	dob	height_cm	weight_kg	club_team_id	club_name	league_name	league_level	c
158023	https://sofifa.co...	L. Messi	Lionel Andrés Mes...	RW, ST, CF	93	93	7.8E7	320000.0	34	1987-06-24	170	72	73.0	Paris Saint-Germain	French Ligue 1	1	
188545	https://sofifa.co...	R. Lewandowski	Robert Lewandowski	ST	92	92	1.195E8	270000.0	32	1988-08-21	185	81	21.0	FC Bayern München	German 1. Bundesliga	1	
20801	https://sofifa.co...	Cristiano Ronaldo	Cristiano Ronaldo...	ST, LW	91	91	4.5E7	270000.0	36	1985-02-05	187	83	11.0	Manchester United	English Premier L...	1	
190871	https://sofifa.co...	Neymar Jr	Neymar da Silva S...	LW, CAM	91	91	1.29E8	270000.0	29	1992-02-05	175	68	73.0	Paris Saint-Germain	French Ligue 1	1	
192985	https://sofifa.co...	K. De Bruyne	Kevin De Bruyne	CM, CAM	91	91	1.255E8	350000.0	30	1991-06-28	181	70	10.0	Manchester City	English Premier L...	1	

only showing top 5 rows

Data

--What are we going to do with the data?

Null value check

Using PySpark's library functions, we check for any NULL or NA values in the dataset and accordingly those rows/columns are dropped depending on the requirement

```
#checking for null values
from pyspark.sql.functions import col,isnan, when, count
players.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in players.columns]).show()
```

sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value_eur	wage_eur	age	dob	height_cm	weight_kg	club_team_id	club_name	league_name	league_level	club_position	club_jersey_number	club_loaned_from
0	0	0	0	0	0	0	74	61	0	0	0	0	61	61	61	61	61	61	18137

We see that in some columns there are lot of missing values whereas in some, they are very little

Checking player columns data type

```
#Checking column data types
players.dtypes
```

```
[('sofifa_id', 'int'),
 ('player_url', 'string'),
 ('short_name', 'string'),
 ('long_name', 'string'),
 ('player_positions', 'string'),
 ('overall', 'int'),
 ('potential', 'int'),
 ('value_eur', 'double'),
 ('wage_eur', 'double'),
 ('age', 'int'),
 ('dob', 'string'),
 ('height_cm', 'int'),
```

Based on the above result, we will separate out the “int” & “double” columns which are not having any NULL or NA values, we will be needing for our analysis and modelling purpose.

```
#creating pyspark dataframe containing all numerical columns (either int or double)

overall_players = players['overall', 'potential', 'age', 'height_cm', 'weight_kg', 'weak_foot', 'skill_moves', 'international_reputation',…
```

```
#creating pyspark dataframe containing all the predictor variables which means overall_players dataframe except 'overall' column

predictors_players = players['potential', 'age', 'height_cm', 'weight_kg', 'weak_foot', 'skill_moves', 'international_reputation',…
```

Above we have created 2 DFs – “overall_players” which contains all numeric columns and “predictors_players” which contains all the numeric predictor variables.

Checking the newly created “overall_players” data frame

```
# Get number of records
print("The data contain %d records." % overall_players.count())


overall_players.show()
```

```
The data contain 19239 records.
+-------+---------+---+---------+---------+---------+-----------+------------------------+------------------+--------------------+--------------------------+-
|overall|potential|age|height_cm|weight_kg|weak_foot|skill_moves|international_reputation|attacking_crossing|attacking_finishing|attacking_heading_accuracy|a
+-------+---------+---+---------+---------+---------+-----------+------------------------+------------------+--------------------+--------------------------+-
|     93|       93| 34|      170|       72|        4|          4|                       5|                85|                  95|                        70|
|     92|       92| 32|      185|       81|        4|          4|                       5|                71|                  95|                        90|
|     91|       91| 36|      187|       83|        4|          5|                       5|                87|                  95|                        90|
|     91|       91| 29|      175|       68|        5|          5|                       5|                85|                  83|                        63|
|     91|       91| 30|      181|       70|        5|          4|                       4|                94|                  82|                        55|
|     91|       93| 28|      188|       87|        3|          1|                       5|                13|                  11|                        15|
```

Null value check in the new data frame. We see that there are no NULL or NA values

```
#Checking for any null values in overall_players pyspark DF

overall_players.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in overall_players.columns]).show()
```

```
+-------+---------+---+---------+---------+---------+-----------+------------------------+------------------+-------------------+--------------------------+-----------------------+----------------+---------------+-----------+-
|overall|potential|age|height_cm|weight_kg|weak_foot|skill_moves|international_reputation|attacking_crossing|attacking_finishing|attacking_heading_accuracy|attacking_short_passing|attacking_volleys|skill_dribbling|skill_curve|s
+-------+---------+---+---------+---------+---------+-----------+------------------------+------------------+-------------------+--------------------------+-----------------------+----------------+---------------+-----------+-
|      0|        0|  0|        0|        0|        0|          0|                       0|                 0|                  0|                         0|                      0|               0|              0|          0|
+-------+---------+---+---------+---------+---------+-----------+------------------------+------------------+-------------------+--------------------------+-----------------------+----------------+---------------+-----------+-
```

Data

--Analysis & Exploration

This is what our “overall_players” pyspark DF looks like now. Note that, we still have 19239 records with 42 columns.

Also, there are no NULL or NA values in “overall_players” data frame.

```python
# Get number of records
print("The data contain %d records." % overall_players.count())


overall_players.show()
```

```
The data contain 19239 records.
```

overall	potential	age	height_cm	weight_kg	weak_foot	skill_moves	international_reputation	attacking_crossing	attacking_finishing	attacking_heading_accuracy	attacking_short_passing	attacking_volleys	skill_dribbling	skill_curve
93	93	34	170	72	4	4	5	85	95	70	91	88	96	93
92	92	32	185	81	4	4	5	71	95	90	85	89	85	79
91	91	36	187	83	4	5	5	87	95	90	80	86	88	81
91	91	29	175	68	5	5	5	85	83	63	86	86	95	88
91	91	30	181	70	5	4	4	94	82	55	94	82	88	85
91	93	28	188	87	3	1	5	13	11	15	43	13	12	13
91	95	22	182	73	4	5	4	78	93	72	85	83	93	80
90	90	35	193	93	4	1	5	15	13	25	60	11	30	14
90	92	29	187	85	4	1	4	18	14	11	61	14	21	18

Using .describe() function to check for statistical parameters of all the numeric columns.

- We see that most of the data lies between 0 – 99 except for few like height, weight, international reputation etc.
- None of the data needs scaling or normalization as the range is pretty similar for all of them

```
1  overall_players.describe().toPandas()
```

	summary	overall	potential	age	height_cm	weight_kg	weak_foot	skill_moves	international_reputation	attacking_crossing	attacking_finishing
0	count	19239	19239	19239	19239	19239	19239	19239	19239	19239	19239
1	mean	65.77218150631529	71.07937002962731	25.210821768283175	181.29970372680492	74.94303238214044	2.9461510473517336	2.352461146629243	1.094183689380945	49.577420863870266	45.89443318259785
2	stddev	6.880231506861786	6.086213101260995	4.748235247092797	6.863179177196197	7.069434064186424	0.6715604780480164	0.7676590344787998	0.37109817520711413	18.03466131695003	19.721022626464077
3	min	47	49	16	155	49	1	1	1	6	2
4	max	93	95	54	206	110	5	5	5	94	95

Correlation matrix

First, we must create an RDD from our “overall_players” before using statistics from pyspark’s MLlib to create a correlation matrix and then use basic pandas' operation to have a data frame having same rows and columns and correlation value.

```
#Creating an RDD for using pyspark's Statistics r
rdd=predictors_players.rdd

rdd.take(2)
```

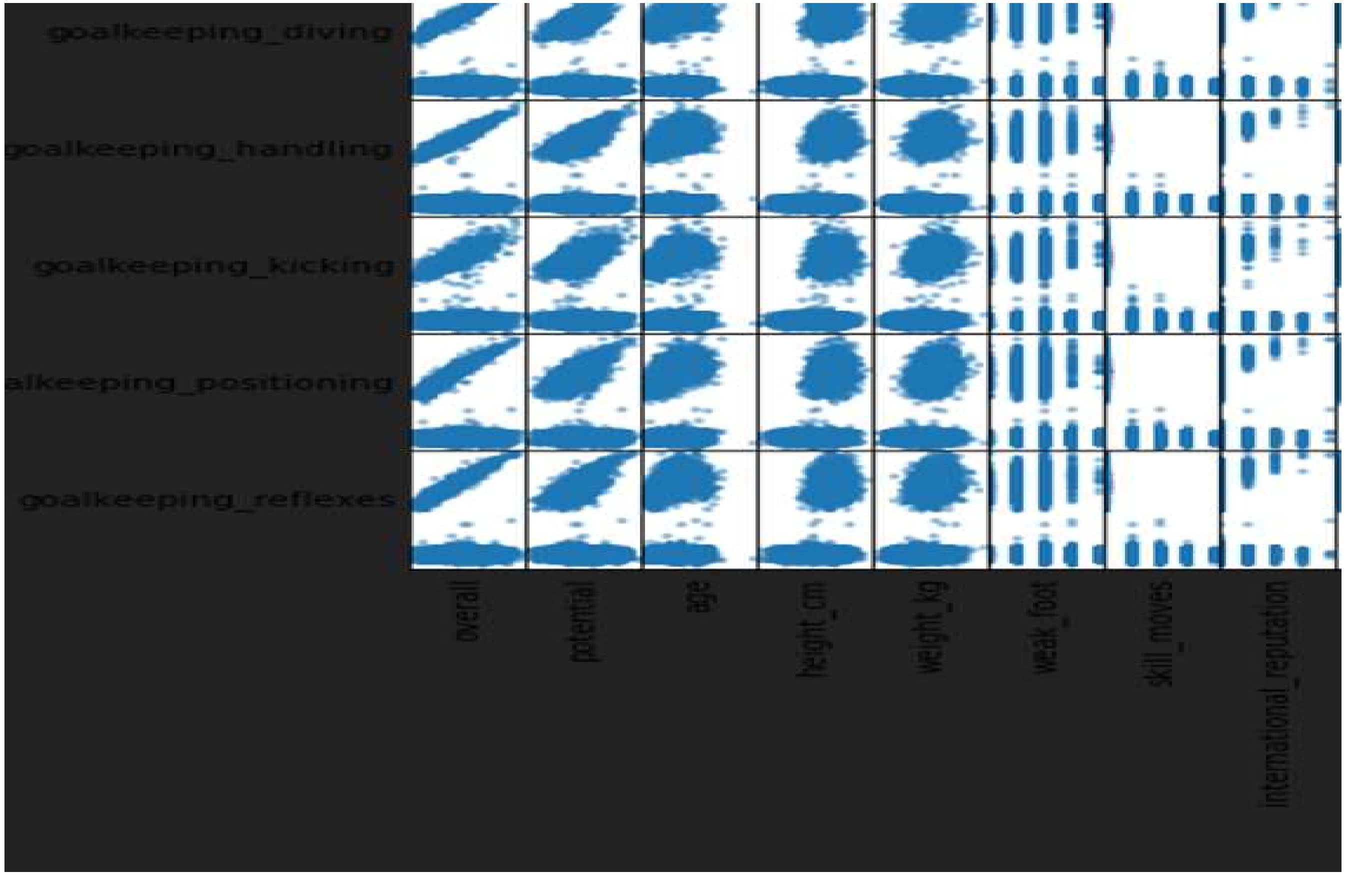
```
features = rdd.map(lambda row: row[0:])

from pyspark.mllib.stat import Statistics

corr_mat=Statistics.corr(features, method="pearson")
```

	potential	age	height_cm	weight_kg	weak_foot	skill_moves	international_reputation	attac
potential	1.000000	-0.264142	0.004403	-0.016912	0.157053	0.283746	0.357283	
age	-0.264142	1.000000	0.083009	0.239444	0.082149	0.074076	0.231927	
height_cm	0.004403	0.083009	1.000000	0.765465	-0.158167	-0.411341	0.042307	
weight_kg	-0.016912	0.239444	0.765465	1.000000	-0.115391	-0.336606	0.088295	
weak_foot	0.157053	0.082149	-0.158167	-0.115391	1.000000	0.344650	0.137155	
skill_moves	0.283746	0.074076	-0.411341	-0.336606	0.344650	1.000000	0.208074	
international_reputation	0.357283	0.231927	0.042307	0.088295	0.137155	0.208074	1.000000	
attacking_crossing	0.243757	0.132175	-0.489842	-0.396283	0.303488	0.721700	0.181019	
attacking_finishing	0.230814	0.088754	-0.374008	-0.290304	0.366527	0.740851	0.167959	
attacking_heading_accuracy	0.192825	0.154516	0.008762	0.039728	0.197269	0.439020	0.148248	

Pandas DF having correlation values



Scatter matrix chart of all numeric features (only few are shown here as original image is too large)

Getting highly correlated columns

```
not_correlated_var_names = correlated_vars_index[correlated_vars_index!=True].index
not_correlated_var_names
```

```
Index(['potential', 'age', 'height_cm', 'weight_kg', 'weak_foot',
       'skill_moves', 'international_reputation', 'attacking_heading_accuracy',
       'movement_reactions', 'movement_balance', 'power_shot_power',
       'power_jumping', 'power_stamina', 'power_strength',
       'mentality_aggression', 'mentality_vision', 'mentality_composure'],
      dtype='object')
```

```
correlated_var_names = correlated_vars_index[correlated_vars_index==True].index
correlated_var_names
```

```
Index(['attacking_crossing', 'attacking_finishing', 'attacking_short_passing',
       'attacking_volleys', 'skill_dribbling', 'skill_curve',
       'skill_fk_accuracy', 'skill_long_passing', 'skill_ball_control',
       'movement_acceleration', 'movement_sprint_speed', 'movement_agility',
       'power_long_shots', 'mentality_interceptions', 'mentality_positioning',
       'mentality_penalties', 'defending_marking_awareness',
       'defending_standing_tackle', 'defending_sliding_tackle',
       'goalkeeping_diving', 'goalkeeping_handling', 'goalkeeping_kicking',
       'goalkeeping_positioning', 'goalkeeping_reflexes'],
      dtype='object')
```

```
corr_reduced_data = overall_players['overall', 'potential', 'age', 'height_cm', 'weight_kg', 'weak_foot',
        'skill_moves', 'international_reputation', 'attacking_heading_accuracy',
        'movement_reactions', 'movement_balance', 'power_shot_power',
        'power_jumping', 'power_stamina', 'power_strength',
        'mentality_aggression', 'mentality_vision', 'mentality_composure']
```

Above we are getting the list of highly correlated columns (abs correlation > 0.8 and < 1) and subsequently list of not highly correlated columns
Also, creating “corr_reduced_data” DF which stores all non highly correlated columns for our use in modelling purpose later on

Modelling

--Linear Regression

As the target column is a continuous variable, we will be using linear regression.
Firstly, by using all the columns and then later using the not highly correlated columns to see what kind of performance we have without the correlated columns.

```
# Import `DenseVector`
from pyspark.ml.linalg import DenseVector

# Define the `input_data`
input_data = overall_players.rdd.map(lambda x: (x[0], DenseVector(x[1:])))

# Replace `df` with the new DataFrame
data = spark.createDataFrame(input_data, ["label", "features"])
```

Splitting dataset into label & features

```
#splitting dataset into train & test group
train_data, test_data = data.randomSplit([0.75, 0.25])

# Check that training set has around 80% of records
training_ratio = train_data.count() / overall_players.count()
print(training_ratio)
```

```
0.7527938042517802
```

Splitting into train & test and verifying size

Modelling

--Using all columns

As mentioned, all the columns are used for modelling and prediction of 'overall' column.
Results –

	Train Data	Test Data	Entire Data
Adjusted R2	0.904	0.907	0.905
RMSE	2.121	2.096	2.115

Above are the adjusted R2 & RMSE results of the trained model. We see –

- Model does not have any underfitting or overfitting problem as the adjusted R2 & RMSE values are same across the entire dataset
- Model can explain 90% of the variance in "overall" column with the help of all the columns used which makes it quite an impressive model

Modelling

--Using all non-highly correlated columns

As mentioned, all the columns which are not highly correlated are used for modelling and prediction of ‘overall’ column. Results –

	Train Data	Test Data	Entire Data
Adjusted R2	0.894	0.891	0.893
RMSE	2.241	2.266	2.247

Above are the adjusted R2 & RMSE results of the trained model. We see –

- Model does not have any underfitting or overfitting problem as the adjusted R2 & RMSE values are same across the entire dataset
- Model can explain ~90% of the variance in “overall” column, which is the same as previous model, with the help of much lesser columns which makes it a brilliant model

Results

--Comparison

Below is the comparison of linear model trained on “all columns” and on “uncorrelated” columns. We are only taking summary results for the entire dataset (and not train or test).

	Entire Data – “All columns”	Entire Data – “Uncorrelated columns”
Adjusted R2	0.905	0. 893
RMSE	2.115	2.247

Above are the adjusted R2 & RMSE results of the trained models on the two datasets. Inferences –

- Model gives the same result even when so many columns have been removed which means “uncorrelated columns” trained model is better
- There is only a slight increase in RMSE but still comparable to the original model
- Time taken for modelling will be lessened significantly now

Results

--Did the end justify the means?

Basically, we were able to train 2 models with the help of entire and reduced dataset and achieve the same result in both dataset. Inferences & path forward –

- As FIFA game is a yearly release, the model can be extended to the latest game edition every year in future or in case of squad update, and we will still be getting quite a splendid result
- We should be using the “uncorrelated columns” dataset trained model for modelling and further ML pipeline purpose

So, at the end yes, the end justify the mean as the entire process of weeding out highly correlated columns served us a model performing the same as initial but with lesser compute time and data requirement. It’s a win!