FIFA

Predict player's positions

Daniel Semerjian

About FIFA

- FIFA is an online soccer video game in which you can play against friends, the computer, or strangers online, The teams and players in the game are realistic and drawn from real life.
- The player's stat’s depends on his position and how well he plays in real life
- FIFA 22 might be the last game in the series to carry the FIFA tag

Different year FIFAS covering

Players positions in FIFA

introduction

My Research question:

❖ can I predict the position of the player based on his stats ?

What made me choose this topic ?

❖ My friends and I play this game all day long

❖FIFA's selection of player positio ns always interests me

❖ I debated with my friend that stats can tell us about the position of the player Therefore, I had extra motivation


CRAWLING

I had to use Selenium to crawl a table properly

I ran on years 2015-2022 to collect 7 years of FIFA players stats

I had to do a time sleep between every page to avoid gettin g blocked

I cut the page url into 3 parts to be able ran on years and pages

```
index = 0 # Using the index, we can determine how many pages were used in which year
for year in range(15,23):
    pages_in_year = int(pages[index]) + 1
    for page in range(1,pages_in_year):
        driver = webdriver.Chrome(PATH)
        driver.get(f"{url_1}{year}{url_2}{page}{url_3}")
        try:
            table = WebDriverWait(driver, 5).until(
                    EC.presence_of_element_located((By.ID, "repTb")) # The class that contains the main table
                )
            body = table.find_element_by_tag_name("tbody") # the tag that contains the table with the stats
            rows = body.find_elements_by_tag_name("tr") # the tag that contains the players stats only !
            for row in rows:
                cols = row.find_elements_by_tag_name("td") # the tag that contains the stats only
                for key,table_item in zip(data.keys(),cols):
                    data[key].append(table_item.text)
        finally:
            driver.quit()
    index += 1
```

The table stored in Id “repTd”, in this Id the table placed in “tbody” tag and the rows of the table stored in “tr” tag, in this “tr” tag the stats stored in “td” tag, So to get the stats in the players' table, I had to run through every layer of this HTML code, step by step.

My Data

	Position	Pace	Shooting	Passing	Dribbling	Defending	Physicality	Height	Agility	Balance	Marking	Positioning	Sprint_Speed	Vision	Finishing
0	CF	93	89	86	96	27	62	169cm | 5'6"	94.0	95.0	25.0	92.0	90.0	90.0	94.0
1	LW	93	93	81	91	32	79	185cm | 6'0"	93.0	63.0	22.0	91.0	94.0	81.0	95.0
2	ST	76	91	81	86	34	86	195cm | 6'4"	86.0	41.0	25.0	86.0	77.0	83.0	91.0
3	RM	93	86	83	92	32	64	180cm | 5'10"	93.0	91.0	29.0	89.0	93.0	84.0	85.0
4	GK	87	85	92	86	58	90	193cm | 6'3"	43.0	35.0	25.0	25.0	61.0	20.0	25.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7822	RM	81	66	68	82	39	56	173cm | 5'8"\nLean (67kg)	86.0	83.0	40.0	68.0	74.0	72.0	70.0
7823	CDM	87	58	66	72	74	81	183cm | 6'0"\nLean (70kg)	81.0	81.0	77.0	68.0	88.0	70.0	62.0
7824	RM	87	71	61	80	52	68	189cm | 6'2"\nHigh & Average+ (79kg)	84.0	73.0	53.0	80.0	89.0	58.0	80.0
7825	LB	89	60	69	75	69	69	176cm | 5'9"\nLean (69kg)	81.0	74.0	71.0	70.0	89.0	65.0	60.0
7826	CB	83	54	67	71	71	83	185cm | 6'1"\nHigh & Average (80kg)	66.0	60.0	67.0	56.0	87.0	65.0	42.0

6719 rows × 15 columns

In order to maximize the result, I took all aspects of the player such as stats related to attack, defense, and playmaking on top of the 6 big stats that combine all of them, and physical stats such as height and skill.

DATA HANDLING

Columns that I dropped

For example The “GK” position was dropped due to his different spatial stats from others

Duplicates

I dropped duplicates with **drop_duplicates** () function

Outliers

My data set had no outliers
to handle since it wa s clean

Data type

My data was mostly not numeric, so I had to convert it with **astype(int)** function

Missing Data

I dropped those missing data because they are of a type that cannot be filled with same values

Normalization

Since most of my data was scaled from 20 to 99 so ,I normalized it to be between 0 and 1

My new data

	Position	Pace	Shooting	Passing	Dribbling	Defending	Physicality	Agility	Balance	Marking	Positioning	Sprint_Speed	Vision	Finishing
0	1	0.205298	0.196468	0.189845	0.211921	0.059603	0.136865	0.638587	0.644728	0.522700	0.541466	0.768049	0.741279	0.876405
1	2	0.198294	0.198294	0.172708	0.194030	0.068230	0.168443	0.666508	0.451049	0.388105	0.512545	0.802185	0.708333	0.847632
2	1	0.167401	0.200441	0.178414	0.189427	0.074890	0.189427	0.652174	0.310606	0.415085	0.495029	0.804094	0.725823	0.829787
3	2	0.206667	0.191111	0.184444	0.204444	0.071111	0.142222	0.659263	0.644433	0.511593	0.542082	0.793651	0.716867	0.820139
4	1	0.181223	0.189956	0.172489	0.192140	0.091703	0.172489	0.637352	0.444215	0.403226	0.529830	0.755403	0.753164	0.867939
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6140	2	0.206633	0.168367	0.173469	0.209184	0.099490	0.142857	0.683987	0.659460	0.578991	0.539683	0.725064	0.750000	0.880078
6141	3	0.198630	0.132420	0.150685	0.164384	0.168950	0.184932	0.733696	0.732954	0.587402	0.614121	0.802773	0.751262	0.887014
6142	2	0.207637	0.169451	0.145585	0.190931	0.124105	0.162291	0.684783	0.594507	0.575372	0.590208	0.811896	0.673497	0.934972
6143	4	0.206497	0.139211	0.160093	0.174014	0.160093	0.160093	0.704348	0.642828	0.580879	0.611111	0.793651	0.667270	0.829787
6144	5	0.193473	0.125874	0.156177	0.165501	0.165501	0.193473	0.606246	0.550576	0.532712	0.543210	0.831899	0.687189	0.645390

6145 rows × 16 columns

This new data has been normalized, and the position column (my target column) represents by numbers the five big positions on the field (striker, wings, midfield, wing defender, and center defender), in addition I added new column that should help me differentiate between wings and midfield positions, I took the stats that identified with high attack profile players, for example Dribbling ,and gave them more weight in the calculation.

Special handle

```python
df[['Player_Height','Delete']] = df['Height'].str.split("cm",n=1,expand=True)
df = df.drop(['Delete', 'Height'],axis=1)
df['Player_Height'] = df['Player_Height'].str.replace(r'\D', '').astype(int) # delete the characters that not number
df['Player_Height'].describe(include='all')
df.drop(df.loc[df["Player_Height"] < 150].index, inplace=True)
# Removed players who are under 150 cm from the table
# Added the new table to the main table and deleted the unnecessary columns
```

I used only the height value here and split it into two parts, leaving only numbers with: **replace(r'\D', ' ').**

```python
df.drop(df.loc[df["Position"] == 'GK'].index, inplace=True)
# Removed the goalkeepers from the table
```

```python
position_dict = {"ST":1 ,"CF":1 ,"RF":2, "RW":2, "LF":2, "LW":2, "LM":2, "RM":2, "CAM":2, "CM":3,
                 "CDM":3, "LB":4, "RB":4, "LWB":4, "RWB":4, "CB":5}
df['Position'] = df['Position'].replace(position_dict).astype(int)
# Combining several positions and giving them values
```

I decided to check every position except GK, so I dropped it. I then gave every position a number that represents it, setting 5 main positions I will attempt to predict

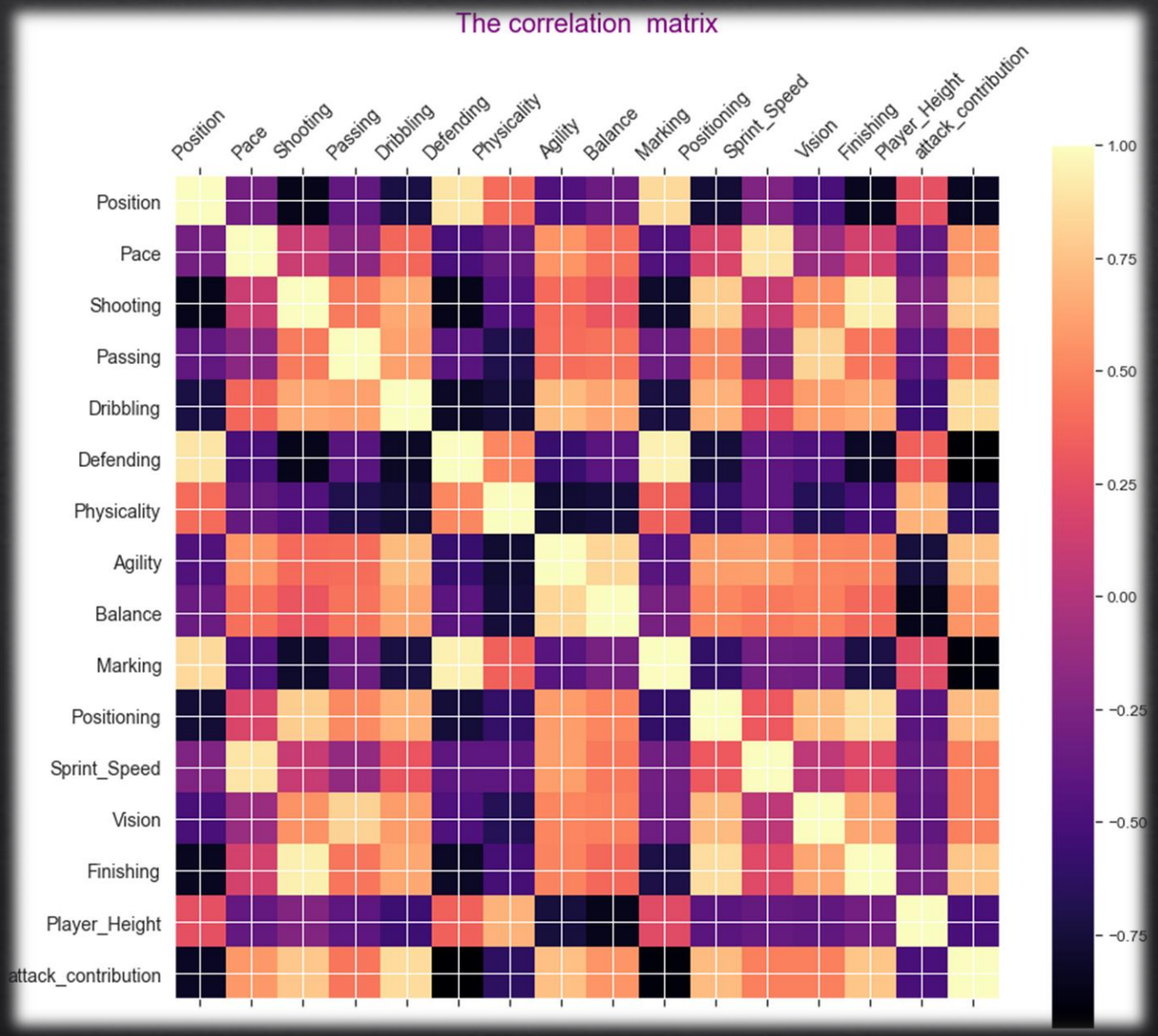
```python
df_normalize1 = df_normalize1.drop(df_normalize1[df_normalize1['Position'] == 2].sample(frac=0.58).index)
df_normalize1 = df_normalize1.drop(df_normalize1[df_normalize1['Position'] == 1].sample(frac=0.31).index)
df_normalize1 = df_normalize1.drop(df_normalize1[df_normalize1['Position'] == 3].sample(frac=0.33).index)
df_normalize1 = df_normalize1.drop(df_normalize1[df_normalize1['Position'] == 5].sample(frac=0.24).index)
```

```python
df_normalize1['Position'].value_counts()
```

```
3    822
1    807
4    799
2    798
5    796
Name: Position, dtype: int64
```

In order to equalize the number of players, I dropped randomly players from positions that had the most players

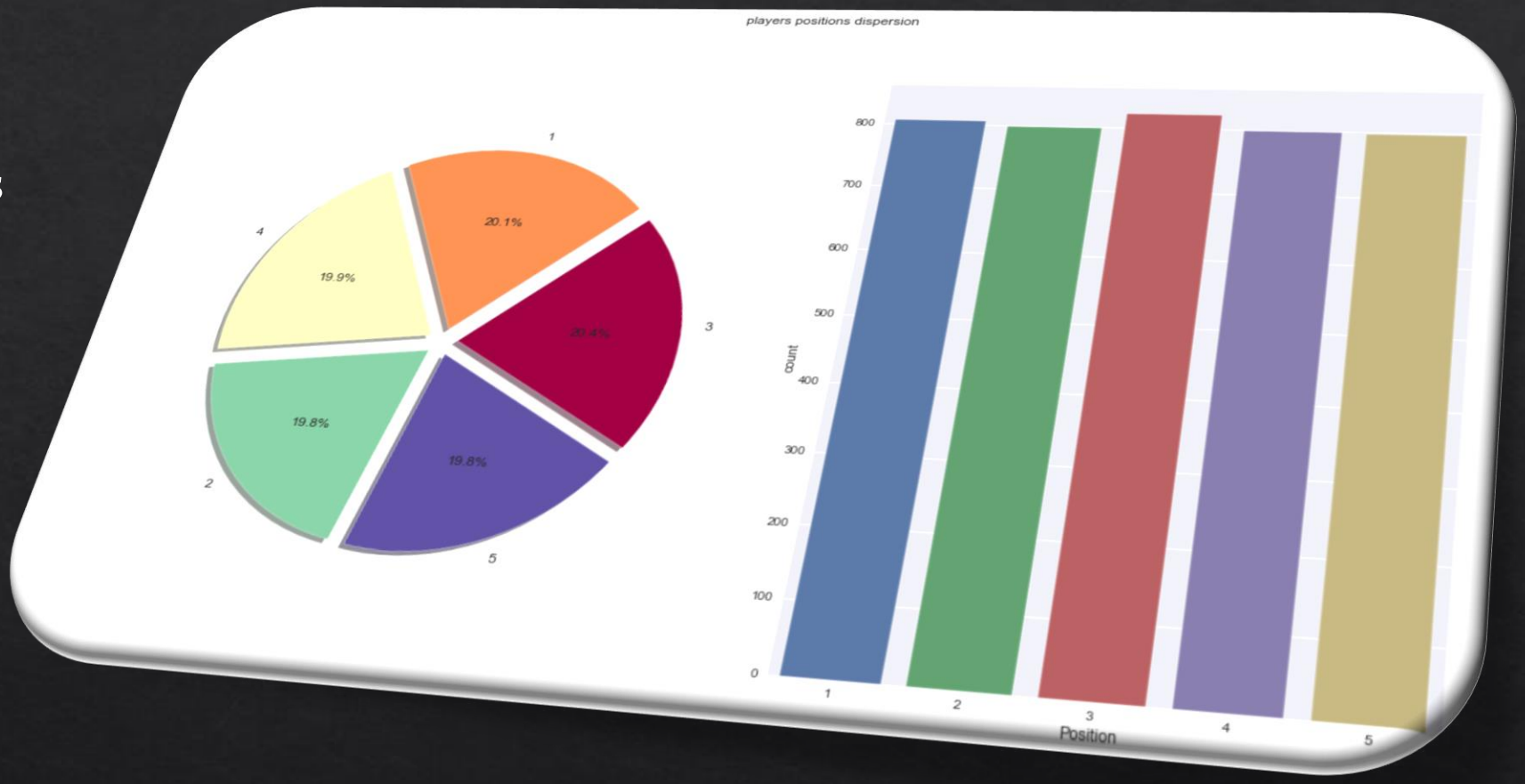
EDA Visualization



My correlation matrix

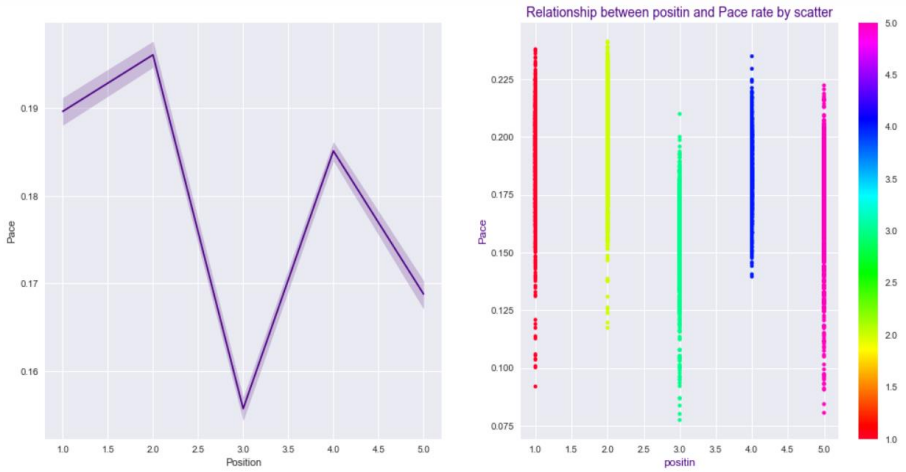
The Players distribution

I had to drop some players, because the number of the players in every position was not equal.



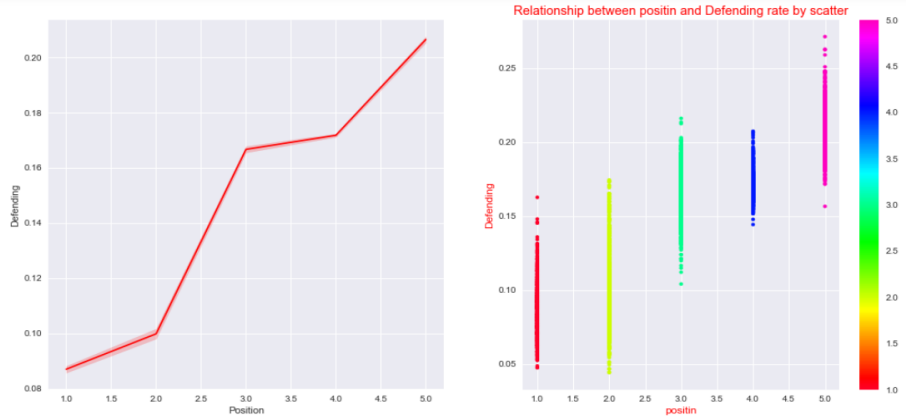
The result

The Relationship between position to other columns



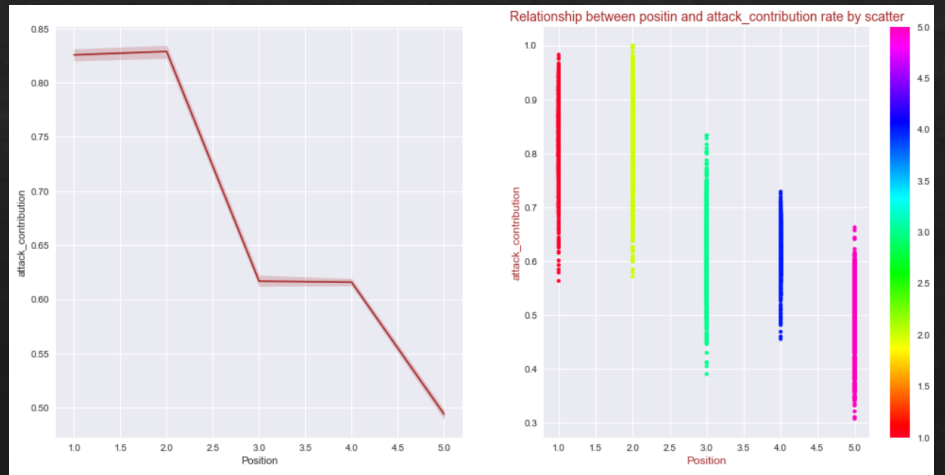
Conclutions:

- player In the defensive position: LB, RB, LWB, RWB, are the fastest
- player In the attack position: RF, RW, LF, LW, CAM are the fastest



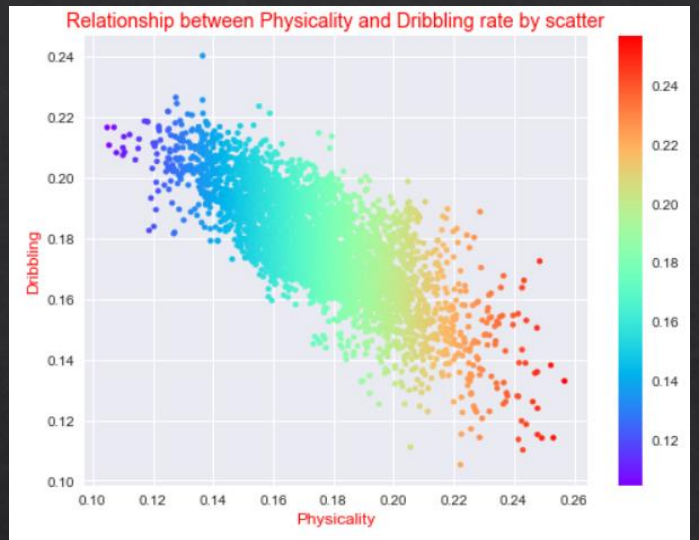
Conclutions:

- When a player's defense attribute is higher, he is more at the back of the field exept except:
- LB
- RB
- LWB
- RWB



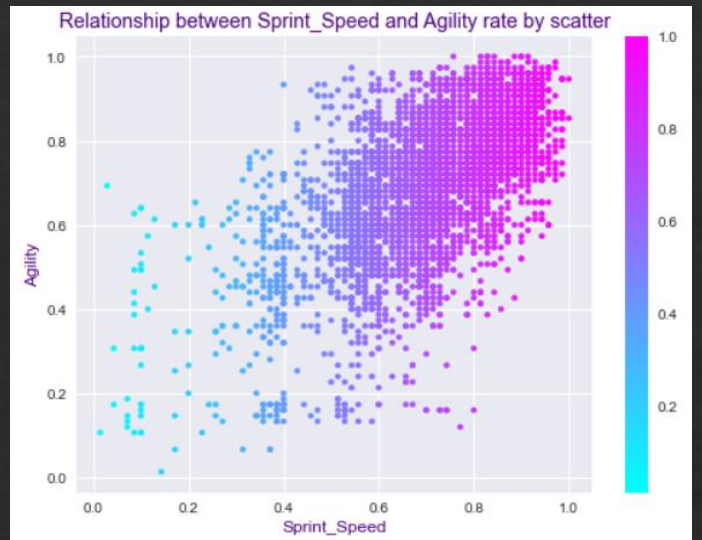
Conclutions:

- Since the player is more in front of the field, he is more likely to contribute to the attack



***Conclutions*:**

- A player with a high Physicality attribute he will Dribbling less well

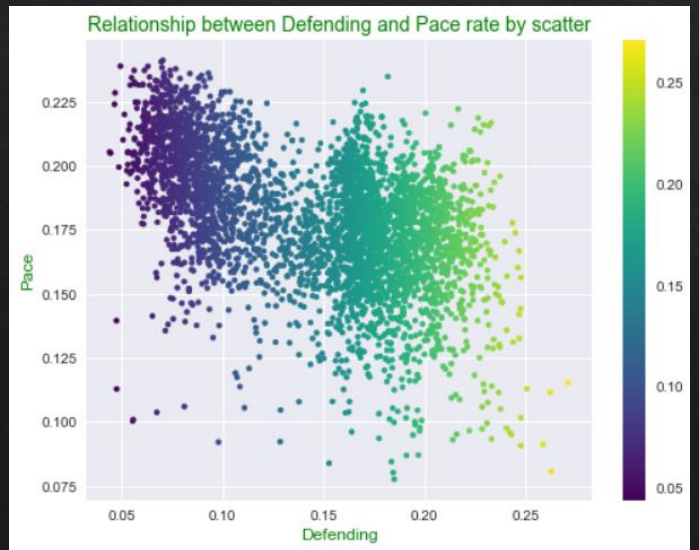


***Conclutions*:**

- Having a high sprint speed will result in greater agility performances

***Conclutions*:**

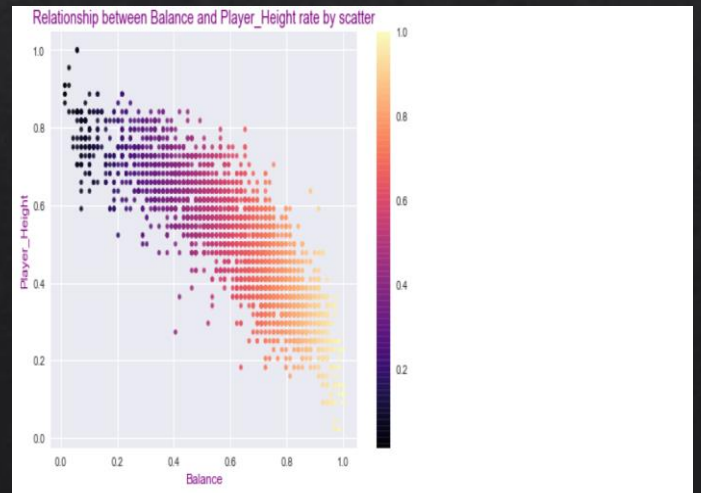
- This graph does not provide a conclusive conclusion



***Conclutions*:**

- This graph does not provide a conclusive conclusion

visualization



***Conclutions*:**

- This graph does not provide a conclusive conclusion, but we can see that if a player is shorter he probably will be balanced

MACHINE LEARNING

My Steps

❖ The first thing I did was split the data to train and test, for to the train I gave 80% of the data and for the test the rest (20%), I set the Y for my target column and the X I set the rest.

❖ In the second step, I tried all 3 models to see which was the most accurate and gave me the best results

❖ I had trouble with the score, at my first attempt I shot 72%, so I went back and played with the data. I averaged some data by each column, normalized The data by the main 6 attributes, added a new column, By using this Feature Engineering, I achieved 86%

The confusion matrix (KNN)

```
array([[127,  21,   0,   0,   0],
       [ 16, 148,  13,  11,   0],
       [  0,   5, 139,  12,   7],
       [  0,   0,   7, 140,   9],
       [  0,   0,   8,   4, 138]], dtype=int64)
```

The confusion matrix (Decision Tree)

```
array([[121,  27,   0,   0,   0],
       [ 32, 133,  14,   9,   0],
       [  0,  10, 128,  18,   7],
       [  0,   4,  14, 132,   6],
       [  0,   0,   8,  10, 132]], dtype=int64)
```

The confusion matrix (SVC)

```
array([[127,  21,   0,   0,   0],
       [ 24, 146,  10,   8,   0],
       [  0,   9, 134,  13,   7],
       [  0,   1,  11, 136,   8],
       [  0,   0,   8,   6, 136]], dtype=int64)
```

SVC:
Support
Vector
Classification

```python
def svc(X_train, y_train, X_test, y_test):
    model = SVC(C=10, kernel='linear')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred
```

```python
def decision_tree(X_train, y_train, X_test, y_test):
    model = tree.DecisionTreeClassifier()
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    return prediction
```

```python
def knn(X_train, y_train, X_test, y_test):
    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.transform(X_test)

    classifier = KNeighborsClassifier(n_neighbors=11, n_jobs=-1)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    return y_pred
```

KNN:
k-nearest
neighbors

Decision Tree

I used supervised machine learning because I had a target column (the players position).
I used KNN, SVC and Decision Tree as my machine learning process.
To check the score I used recall, f1, precision and accuracy scores.

	model	recall	f1	accuracy_s	precision
0	SVC	0.847050	0.845581	0.843478	0.844604
1	Decision Tree	0.810274	0.807907	0.804969	0.808226
2	KNN	0.863108	0.861545	0.859627	0.861090

we can see that KNN gave us the best results 86%

REFERENCES

References

- https://www.futbin.com/
- https://stackoverflow.com/
- https://www.geeksforgeeks.org/
- https://pandas.pydata.org/docs/
- https://matplotlib.org/stable/tutorials/introductory/pyplot.html
- https://unsplash.com/
- https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/#32.-Pie-Chart
- https://fifauteam.com/fifa-21-positions/
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning