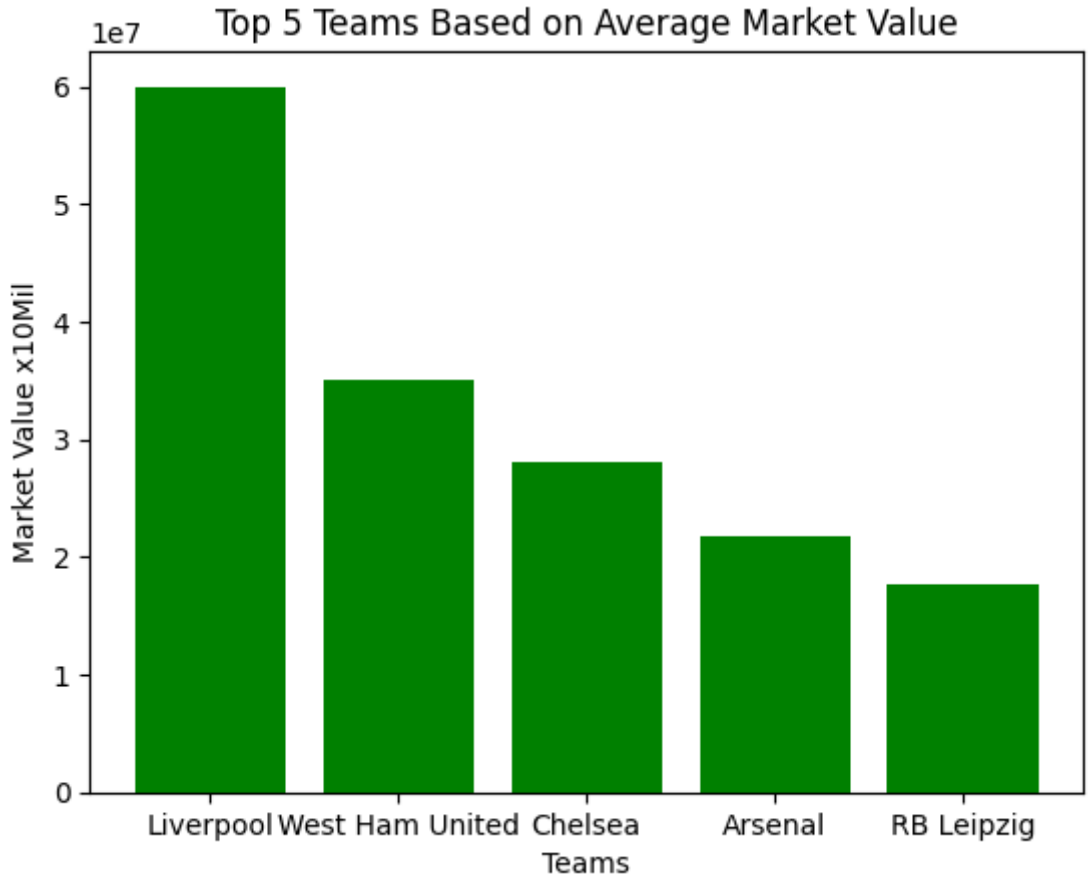
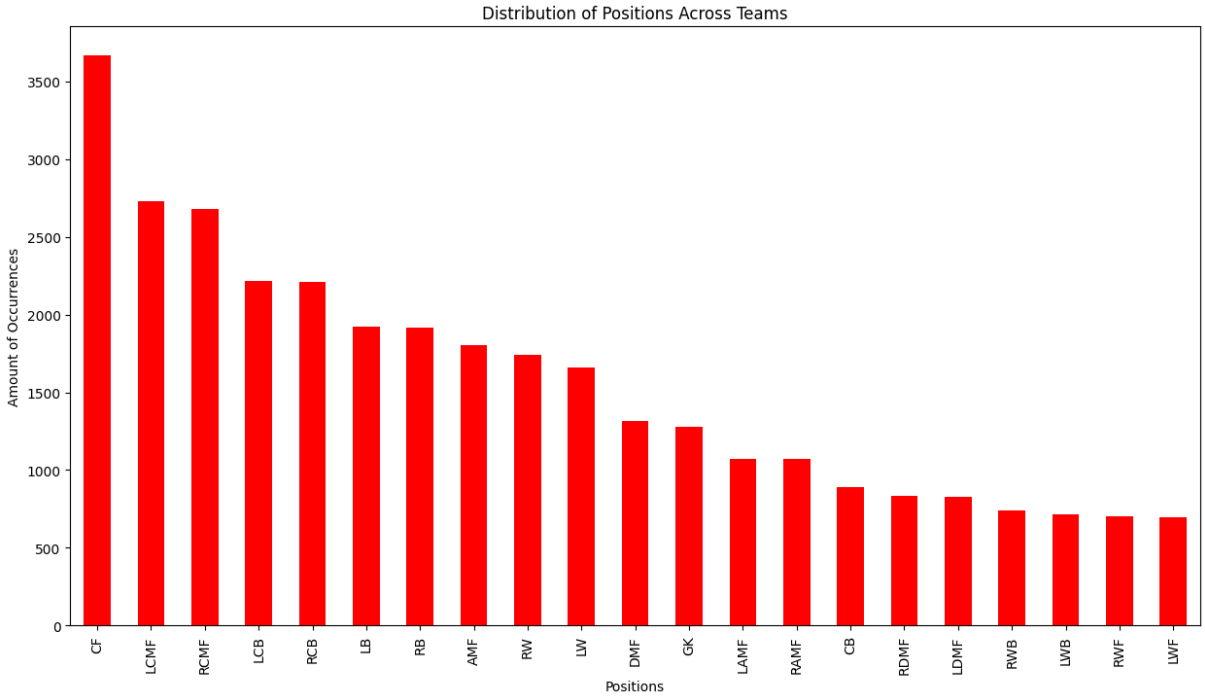
Jason, 1B, Feature analysis and Simple Classification Model.

Analysis:

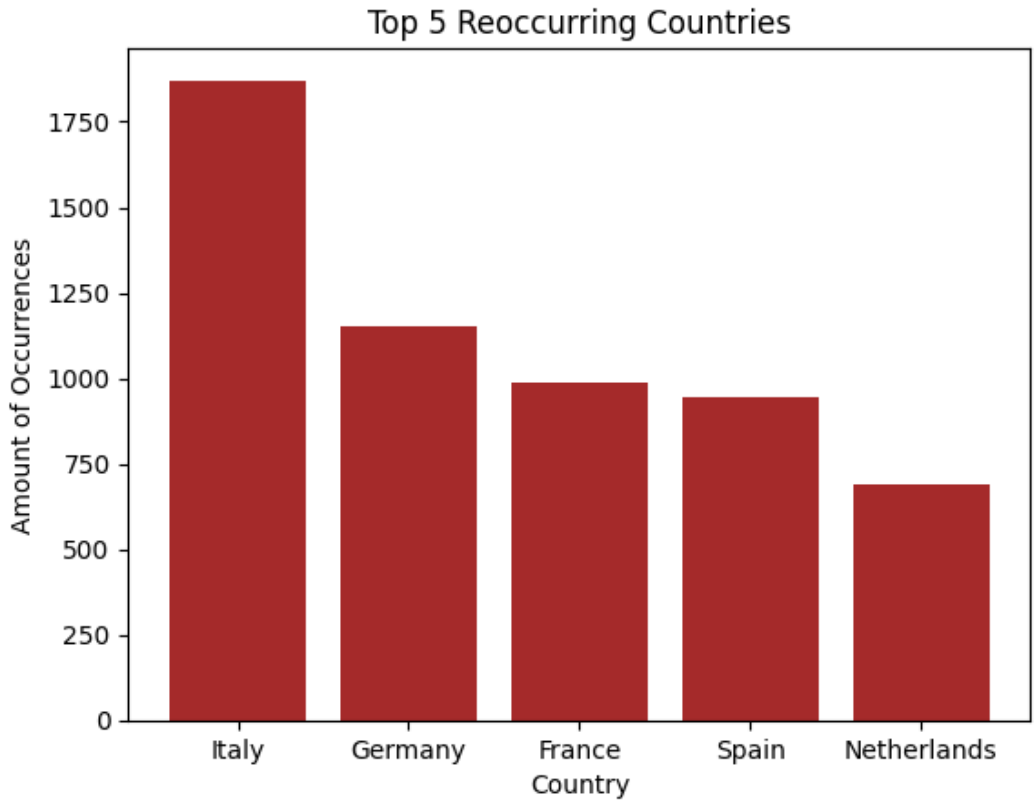
Before starting on the Classification Model, we first had to do some analysis into the features in the dataset. Below are some graphs I used for analysis and their interpretations.



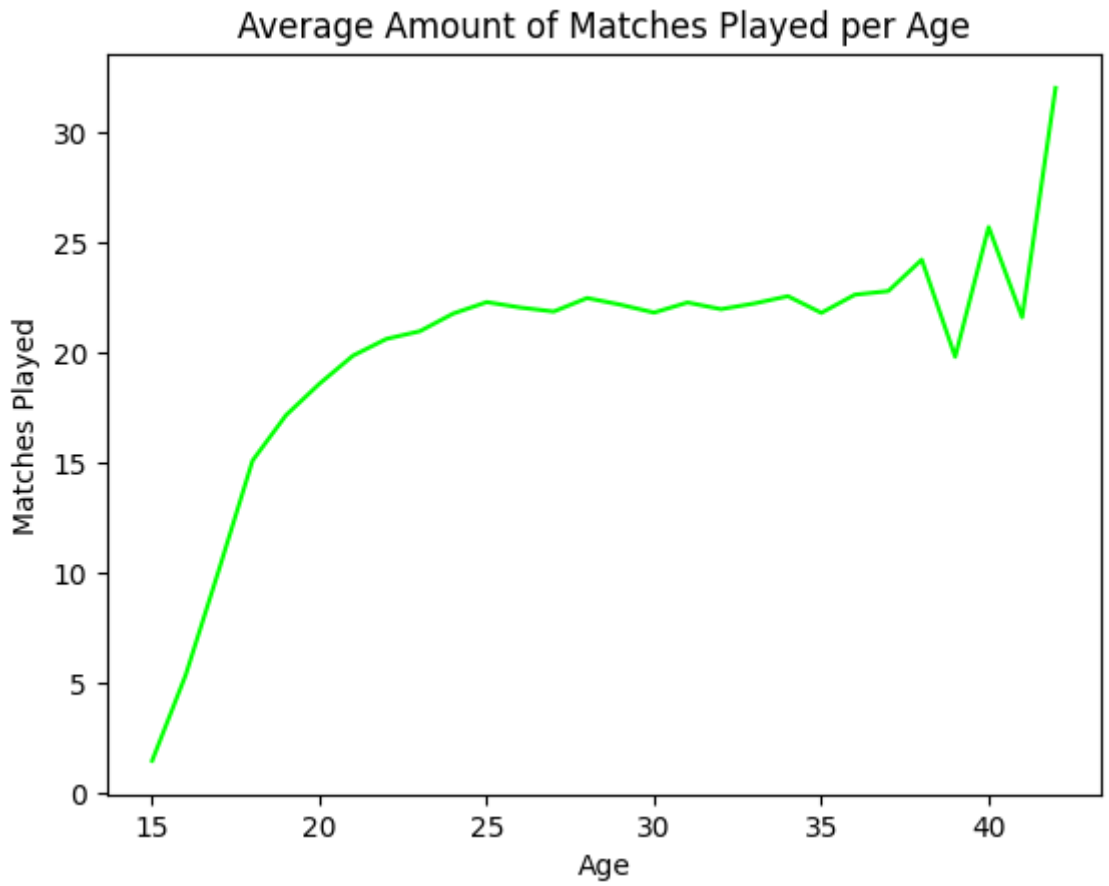
Here we visualize the top 5 teams based on Average Market Value. We can see here that the average Market Value of Liverpool is very high compared to the rest, followed is West Ham United with a Market Value which seems to be around 60% of Liverpool's.



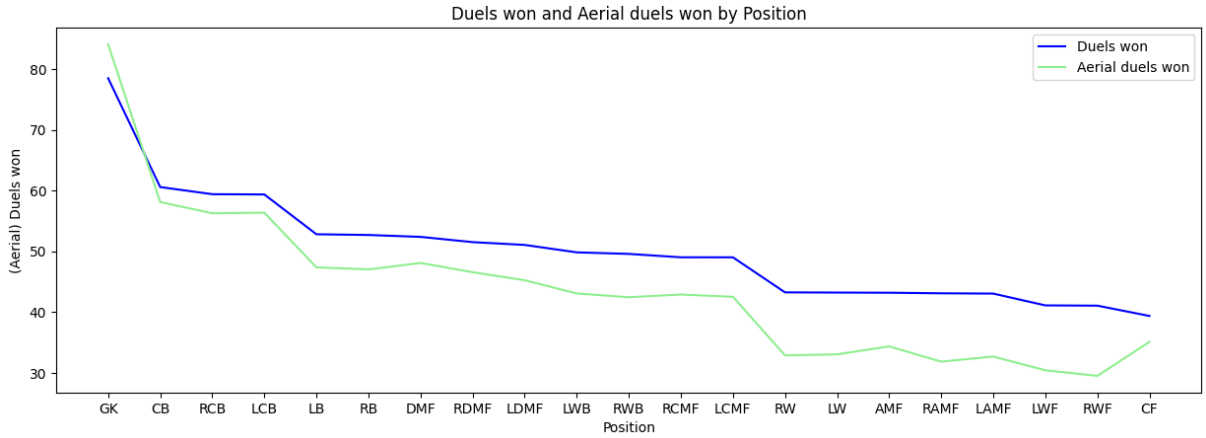
Next we visualized the distribution of Positions between players. Here we can see again that CF is the most popular position by far.



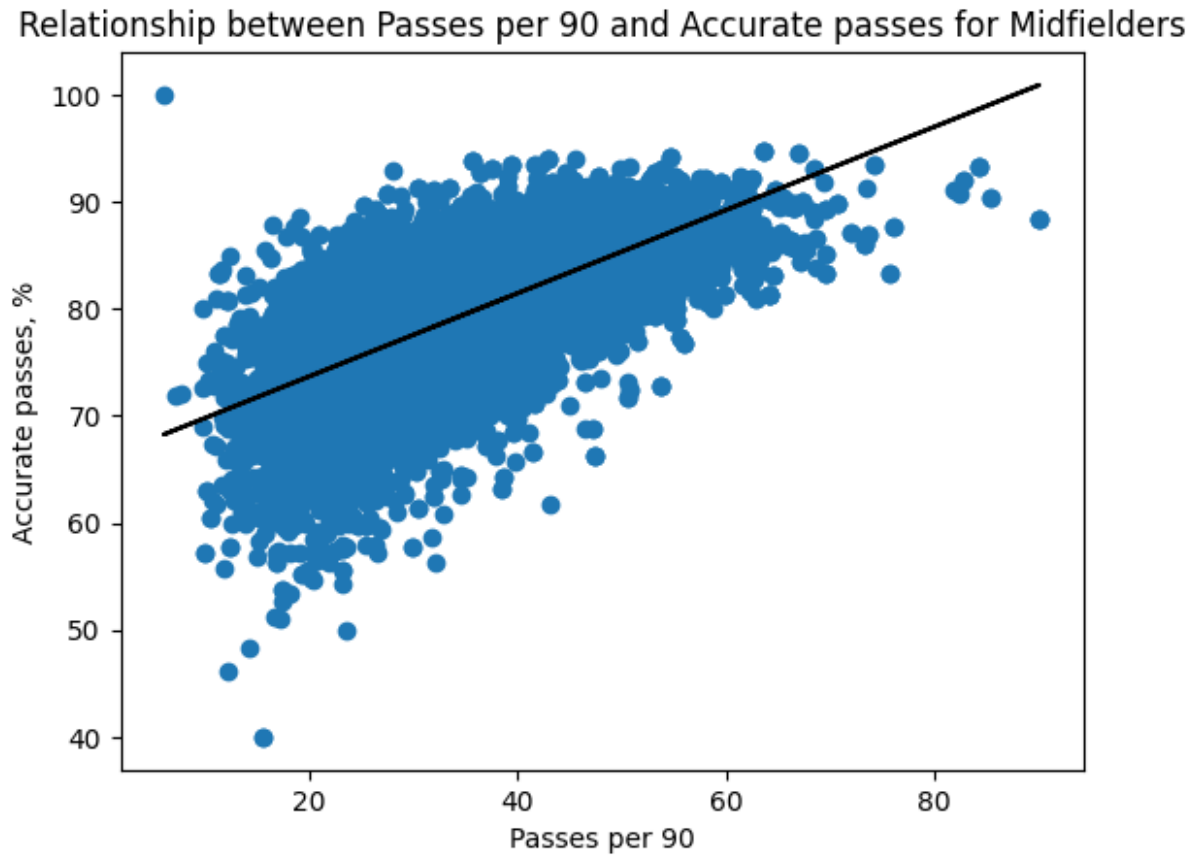
I chose to only visualize the top 5, just so the graph will stay readable and won't overflow you with information. In this visual we can see that the country on first place is Italy, the amount of players from Italy heavily outweigh the other countries.



we can see that the younger a player is, the least matches they play, but this changes after around the age of 23: from this point onwards the line seems to be staying around the same value. There seems to be random peaks of data after the age of 35, but this is due to the lack of players in that age group in the dataset.



How do duels won and aerial duels won vary by position? Looking at the visual, we can keep the conclusion made earlier: the more defensive positions have the higher average amount of duels won and both datapoints have about the same distribution.



I can see that there is a really slight correlation to the two variables. There has to be a different variable which also has impact on this correlation, but it needs some extra future research.

Machine Learning:

First we started making a basic model without changing (preprocessing) the data too much, which gave the following results per type of model:

Linear Regression:

Linear regression

First off, we will train the Linear Regression model. The scores seem to be very low, but I will look further into the performances later on.

```
# Create the Linear Regression Model
lr = LinearRegression()
# Train the model
lr.fit(X_train_lin, y_train_lin)

# Predict using the test set
y_pred_lin = lr.predict(X_test_lin)

# Calculate and print the score.
print("Score testing data:", lr.score(X_test_lin, y_test_lin))
print("Score training data", lr.score(X_train_lin, y_train_lin))
```

[57] Python

```
Score testing data: 0.04626023456830519
Score training data 0.05587548583185742
```

Logistic Regression:

Logistic regression

Next, we will train the Logistic Regression model, which seems to have a much higher score and doesn't underfit nor overfit, this score will increase in our final model and we will evaluate the scores later on.

```python
# Create Logistic Regression Model.
log_reg = LogisticRegression(max_iter=100000)

# Train the model
log_reg.fit(X_train, y_train)

# Make a prediction
y_pred_log = log_reg.predict(X_test)

# Calculate and print the score
print("Score testing data:", log_reg.score(X_test, y_test))
print("Score training data:", log_reg.score(X_train, y_train))
```

[58] Python

```
Score testing data: 0.5062684741616553
Score training data: 0.5086493098025511
```

Random Forest:

Tree-based model

Next is the Random Forest Classifier, which shows to have a slightly lower score on both testing and training data compared to Logistic Regression. The scores will be evaluated further later on.

```python
# Create the Model
clf = RandomForestClassifier(max_depth=2, min_samples_split=2, n_estimators=50)

# Train model
clf.fit(X_train, y_train)

# Predict the result
y_pred_clf = clf.predict(X_test)

# Calculate and print out the score.
print("Score testing data:", clf.score(X_test, y_test))
print("Score training data:", clf.score(X_train, y_train))
```

[59] Python

```
Score testing data: 0.49261033533788606
Score training data: 0.49515114450463044
```

Gradient Boosting Trees:

Gradient Boosting Trees

Now we will look at Gradient Boosting, which seems to have a much higher score and seems to be the bette model at this moment, but it also overfits the data quite a bit. We will compare the scores later to choose which model is better.

```python
# Initiate the gradient booster classifier
gb = XGBClassifier(max_depth=2, n_estimators=200)

# Train the model.
gb.fit(X_train, y_train)

# Predict the values
y_pred_gb = gb.predict(X_test)

# Print score
print("Score testng data:", gb.score(X_test, y_test))
print("Score training data:", gb.score(X_train, y_train))
```

[60] Python

```
Score testng data: 0.5475486698603608
Score training data: 0.5764896033548838
```

To improve these models I applied Feature and Correlation Analysis per each position:

Goalkeeper:

	Market value	Bin	Contract expires	Matches played	Minutes played	Height	Weight	Fouls suffered per 90	Accurate passes, %	Conceded goals	Conceded goals per 90	Shots against	Shots against per 90	Clean sheets	Save rate, %
Market value	1.000000	0.485393	0.280070	0.246147	0.251751	0.129378	0.149024	-0.123180	0.106741	0.129953	-0.171538	0.166334	-0.159537	0.294595	0.100221
Bin	0.485393	1.000000	0.346805	0.400275	0.406892	0.295114	0.349269	-0.152968	0.111112	0.299508	-0.224811	0.328260	-0.208320	0.399464	0.125144
Contract expires	0.280070	0.346805	1.000000	0.179100	0.182446	0.089198	0.085999	-0.089809	0.086586	0.113345	-0.111213	0.132504	-0.112987	0.183654	0.050338
Matches played	0.246147	0.400275	0.179100	1.000000	0.999226	0.152478	0.196590	-0.041415	0.050813	0.852166	-0.295991	0.923829	-0.138688	0.819337	0.314804
Minutes played	0.251751	0.406892	0.182446	0.999226	1.000000	0.155224	0.198708	-0.045639	0.050283	0.851170	-0.298169	0.922518	-0.143468	0.821155	0.313155
Height	0.129378	0.295114	0.089198	0.152478	0.155224	1.000000	0.793160	-0.047228	0.025066	0.098577	-0.141205	0.118787	-0.118062	0.137625	0.095337
Weight	0.149024	0.349269	0.085999	0.196590	0.198708	0.793160	1.000000	-0.086972	0.048135	0.136675	-0.161444	0.156020	-0.152262	0.178766	0.083164
Fouls suffered per 90	-0.123180	-0.152968	-0.089809	-0.041415	-0.045639	-0.047228	-0.086972	1.000000	-0.069450	-0.036837	0.030372	-0.029253	0.045259	-0.048313	-0.041529
Accurate passes, %	0.106741	0.111112	0.086586	0.050813	0.050283	0.025066	0.048135	-0.069450	1.000000	-0.013171	-0.106251	-0.009762	-0.159998	0.075373	-0.020990
Conceded goals	0.129953	0.299508	0.113345	0.852166	0.851170	0.098577	0.136675	-0.036837	-0.013171	1.000000	0.103868	0.948590	0.187983	0.462489	0.059063
Conceded goals per 90	-0.171538	-0.224811	-0.111213	-0.295991	-0.298169	-0.141205	-0.161444	0.030372	-0.106251	0.103868	1.000000	-0.077042	0.748255	-0.527784	-0.596876
Shots against	0.166334	0.328260	0.132504	0.923829	0.922518	0.118787	0.156020	-0.029253	-0.009762	0.948590	-0.077042	1.000000	0.150750	0.617803	0.273749
Shots against per 90	-0.159537	-0.208320	-0.112987	-0.138688	-0.143468	-0.118062	-0.152262	0.045259	-0.159998	0.187983	0.748255	0.150750	1.000000	-0.363851	0.005323
Clean sheets	0.294595	0.399464	0.183654	0.819337	0.821155	0.137625	0.178766	-0.048313	0.075373	0.462489	-0.527784	0.617803	-0.363851	1.000000	0.439228
Save rate, %	0.100221	0.125144	0.050338	0.314804	0.313155	0.095337	0.083164	-0.041529	-0.020990	0.059063	-0.596876	0.273749	0.005323	0.439228	1.000000

Attackers:

	Market value	Bin	Contract expires	Matches played	Minutes played	Goals	xG	Assists	xA	Height	Weight	Goals per 90	Non-penalty goals	xG per 90	Shots	Shots per 90	Touches in box per 90	Received passes per 90	xA per 90	Shot assists per 90	Smart passes per 90	Key passes per 90	Deep completions per 90
Market value	1.000000	0.372818	0.312464	0.089086	0.139652	0.211853	0.203589	0.203478	0.204146	0.119952	0.139238	0.160859	0.103088	0.161858	0.190146	0.150992	0.181620	0.125257	0.167710	0.157743	0.119608	0.182076	0.202812
Bin	0.372818	1.000000	0.375802	0.271376	0.320970	0.279840	0.304708	0.253135	0.289079	0.372500	0.413473	0.150008	0.226966	0.174596	0.328198	0.195828	0.177551	0.206083	0.172716	0.194394	0.197304	0.200511	0.212367
Contract expires	0.312464	0.375802	1.000000	0.059958	0.100616	0.140426	0.137223	0.102009	0.104543	0.133754	0.145009	0.112083	0.103708	0.127124	0.139079	0.132377	0.145909	0.049561	0.075046	0.063658	0.063405	0.100688	0.114901
Matches played	0.089086	0.271376	0.059958	1.000000	0.865626	0.546970	0.587576	0.463566	0.539093	0.182733	0.188609	0.147596	0.482658	0.104821	0.693514	0.133908	0.090785	0.059503	0.111275	0.128055	0.108661	0.121366	0.076481
Minutes played	0.139652	0.320970	0.100616	0.865626	1.000000	0.673994	0.709176	0.557688	0.657731	0.176292	0.186563	0.193551	0.526220	0.131347	0.820120	0.153291	0.089460	0.119729	0.145408	0.166714	0.160038	0.148201	0.120988
Goals	0.211853	0.279840	0.140426	0.546970	0.673994	1.000000	0.897928	0.425201	0.440750	0.136303	0.164964	0.703596	0.573379	0.549062	0.816706	0.442979	0.369799	0.048840	0.092902	0.078946	0.128884	0.115733	0.155315
xG	0.203589	0.304708	0.137223	0.587576	0.709176	0.897928	1.000000	0.417959	0.421455	0.178861	0.211633	0.549356	0.515308	0.656349	0.877330	0.505630	0.449794	0.112156	0.059345	0.049091	0.105023	0.098658	0.139199
Assists	0.203478	0.253135	0.102009	0.463566	0.557688	0.425201	0.417959	1.000000	0.773599	0.100580	0.110815	0.149468	0.321887	0.097213	0.534364	0.181263	0.153521	0.265286	0.501217	0.456727	0.318737	0.459724	0.354835
xA	0.204146	0.289079	0.104543	0.539093	0.657731	0.440750	0.421455	0.773599	1.000000	0.115846	0.112778	0.124538	0.367078	0.041361	0.600852	0.183539	0.127034	0.380375	0.692108	0.640256	0.377975	0.523447	0.429273
Height	0.119952	0.372500	0.133754	0.182733	0.176292	0.136303	0.178861	0.100580	0.115846	1.000000	0.801558	0.089817	0.141006	0.151592	0.181096	0.160723	0.141438	0.064692	0.079042	0.108626	0.094987	0.107543	0.098068
Weight	0.139238	0.413473	0.145009	0.188609	0.186563	0.164964	0.211633	0.110815	0.112778	0.801558	1.000000	0.111272	0.150578	0.175063	0.204061	0.170980	0.155524	0.063082	0.055736	0.090997	0.097280	0.093037	0.098703
Goals per 90	0.160859	0.150008	0.112083	0.147596	0.193551	0.703596	0.549356	0.149468	0.124538	0.089817	0.111372	1.000000	0.519960	0.715323	0.415825	0.543367	0.482417	0.117831	0.035285	0.002758	0.066391	0.067359	0.130843
Non-penalty goals	0.103088	0.226966	0.103708	0.482658	0.526220	0.573379	0.515308	0.321887	0.367078	0.141006	0.150578	0.519960	1.000000	0.361346	0.555356	0.344975	0.296271	0.013676	0.107421	0.099598	0.115185	0.120164	0.136461
xG per 90	0.161858	0.174596	0.127124	0.104821	0.131347	0.549062	0.656349	0.097213	0.041361	0.151592	0.175063	0.715323	0.361346	1.000000	0.442527	0.737643	0.686809	0.239951	0.011553	0.036821	0.026320	0.048275	0.129819
Shots	0.190146	0.328198	0.139079	0.693514	0.820120	0.816706	0.877330	0.534364	0.600852	0.181096	0.204061	0.415825	0.555356	0.442527	1.000000	0.575527	0.363481	0.070836	0.164316	0.264951	0.221206	0.189356	0.218767
Shots per 90	0.150992	0.195828	0.132377	0.133908	0.153291	0.442979	0.505630	0.181263	0.183539	0.160723	0.170980	0.543367	0.344975	0.737643	0.575527	1.000000	0.650589	0.008991	0.175224	0.195270	0.207111	0.194170	0.268049
Touches in box per 90	0.181620	0.177551	0.145909	0.090785	0.089460	0.369799	0.449794	0.153521	0.127034	0.141438	0.155524	0.482417	0.296271	0.686809	0.363481	0.650589	1.000000	0.143077	0.171636	0.143635	0.058867	0.279140	0.381619
Received passes per 90	0.125257	0.206083	0.049561	0.059503	0.119729	0.048840	0.112156	0.265286	0.380375	0.064692	0.063082	0.117831	0.013676	0.239951	0.070836	0.008991	0.143077	1.000000	0.422185	0.517891	0.478340	0.359144	0.470029
xA per 90	0.167710	0.172716	0.075046	0.111275	0.145408	0.092902	0.059345	0.501217	0.692108	0.079042	0.055736	0.035285	0.107421	0.011553	0.184316	0.175224	0.171636	0.422185	1.000000	0.834879	0.394037	0.731063	0.532535
Shot assists per 90	0.157743	0.194394	0.063658	0.128055	0.166714	0.078946	0.049091	0.456727	0.640256	0.108626	0.090997	0.002758	0.099598	0.036821	0.204951	0.195270	0.143635	0.517891	0.834879	1.000000	0.453962	0.701470	0.593878
Smart passes per 90	0.119608	0.197304	0.063405	0.108661	0.160038	0.128884	0.105023	0.318737	0.377975	0.094987	0.097280	0.066391	0.115185	0.026320	0.221206	0.207111	0.058867	0.478340	0.394037	0.453962	1.000000	0.371051	0.538050
Key passes per 90	0.182076	0.200511	0.100688	0.121366	0.148201	0.115733	0.098658	0.459724	0.523447	0.107543	0.093037	0.067359	0.120164	0.048275	0.189356	0.194170	0.279140	0.359144	0.731063	0.701470	0.371051	1.000000	0.559108
Deep completions per 90	0.202812	0.212367	0.114901	0.076481	0.120988	0.155315	0.139199	0.354835	0.429273	0.098068	0.098703	0.130843	0.136461	0.129819	0.218767	0.268049	0.381619	0.470029	0.532535	0.593878	0.538050	0.559108	1.000000

Defenders and MidFielders (combination)

	Market value	Bin	Age	Contract expires	Matches played	Minutes played	Goals	xG	Assists	xA	Height	Weight	Non-penalty goals	Shots	Received passes per 90	Passes per 90	Accurate passes, %	Accurate forward passes, %	Back passes per 90	Short / medium passes per 90	Accurate passes to final third, %	Accurate progressive passes, %
Market value	1.000000	0.381022	0.004059	0.296088	0.111956	0.137036	0.189235	0.193883	0.170730	0.177988	0.119174	0.128867	0.157061	0.184213	0.148504	0.116294	0.094560	0.101550	0.109705	0.131286	0.123943	0.137270
Bin	0.381022	1.000000	0.196084	0.354195	0.256104	0.282956	0.199018	0.228567	0.193725	0.227376	0.364586	0.394988	0.192011	0.248097	0.211798	0.169604	0.161617	0.171045	0.134680	0.183557	0.199179	0.209726
Age	0.004059	0.196094	1.000000	-0.137757	0.164463	0.202064	0.051473	0.081999	0.071904	0.096440	0.355556	0.379511	0.038030	0.068376	0.149873	0.164749	0.119757	0.140702	-0.032452	0.140572	0.073801	0.153980
Contract expires	0.296088	0.354195	-0.137757	1.000000	0.058174	0.076958	0.115486	0.111756	0.095681	0.099221	0.110991	0.114275	0.110038	0.116582	0.108737	0.077412	0.082931	0.070607	0.101685	0.093588	0.099417	0.085371
Matches played	0.111956	0.256104	0.164463	0.058174	1.000000	0.891556	0.411464	0.475354	0.394526	0.469007	0.152582	0.148097	0.411432	0.561726	0.029943	0.038214	-0.002477	0.015011	0.069674	0.027430	0.087492	0.123065
Minutes played	0.137036	0.282956	0.202064	0.076958	0.891556	1.000000	0.371149	0.431284	0.371511	0.438914	0.161838	0.172497	0.355490	0.499948	0.159853	0.197265	0.127490	0.140358	-0.032124	0.167663	0.087917	0.116099
Goals	0.189235	0.199018	0.051473	0.115486	0.411464	0.371149	1.000000	0.867367	0.462747	0.513803	0.057927	0.051220	0.822047	0.790333	-0.104952	-0.171859	-0.128311	-0.107980	0.167811	-0.144616	0.056143	0.166452
xG	0.193883	0.228567	0.081999	0.111756	0.475354	0.431284	0.867367	1.000000	0.476173	0.531058	0.086130	0.078467	0.703802	0.874086	-0.135051	-0.203744	-0.139400	-0.120764	0.151890	-0.169798	0.055183	0.167727
Assists	0.170730	0.193725	0.071904	0.095681	0.394526	0.371511	0.462747	0.476173	1.000000	0.789274	0.049550	0.036537	0.402023	0.562508	0.028373	-0.029707	-0.165312	-0.091902	0.277068	-0.050834	0.068120	0.234857
xA	0.177988	0.227376	0.096440	0.099221	0.469007	0.438914	0.513803	0.531058	0.789274	1.000000	0.065004	0.041667	0.456337	0.656878	0.027190	-0.041858	-0.221725	-0.120091	0.338538	-0.077697	0.078119	0.202391
Height	0.119174	0.364586	0.355556	0.110991	0.152582	0.161838	0.057927	0.086130	0.049550	0.065004	1.000000	0.835063	0.068556	0.084834	0.127150	0.118599	0.120138	0.124454	0.012994	0.118484	0.110768	0.116234
Weight	0.128867	0.394988	0.379511	0.114275	0.148097	0.172497	0.051220	0.078467	0.036537	0.041667	0.835063	1.000000	0.058980	0.068918	0.153813	0.151206	0.158529	0.147724	-0.017500	0.149673	0.106906	0.111926
Non-penalty goals	0.157061	0.192011	0.038030	0.110038	0.411432	0.355490	0.822047	0.703802	0.402023	0.456337	0.068556	0.058980	1.000000	0.690361	-0.108728	-0.172187	-0.125279	-0.103899	0.166146	-0.144371	0.058309	0.152854
Shots	0.184213	0.248097	0.068376	0.116582	0.561726	0.499948	0.790333	0.874086	0.562508	0.656878	0.084834	0.068918	0.690361	1.000000	-0.092493	-0.159995	-0.175655	-0.146135	0.249943	-0.143208	0.084530	0.233912
Received passes per 90	0.148504	0.211798	0.149873	0.108737	0.029943	0.159853	-0.104952	-0.135051	0.028373	0.027190	0.127150	0.153813	-0.108728	-0.092493	1.000000	0.958321	0.625670	0.572173	0.271224	0.948436	0.359158	0.335907
Passes per 90	0.116294	0.169604	0.164749	0.077412	0.038214	0.197265	-0.171859	-0.203744	-0.029707	-0.041858	0.118599	0.151206	-0.172187	-0.159995	0.958321	1.000000	0.651067	0.586340	0.159711	0.975833	0.318586	0.278130
Accurate passes, %	0.094560	0.161617	0.119757	0.082931	-0.002477	0.127490	-0.128311	-0.139400	-0.165312	-0.221725	0.120138	0.158529	-0.125279	-0.175655	0.625670	0.651067	1.000000	0.802486	0.076570	0.735076	0.492844	0.239692
Accurate forward passes, %	0.101550	0.171045	0.140702	0.070607	0.015011	0.140358	-0.107980	-0.120764	-0.091902	-0.120091	0.124454	0.147724	-0.103899	-0.146135	0.572173	0.586340	0.802486	1.000000	0.006347	0.635328	0.561493	0.388892
Back passes per 90	0.109705	0.134680	-0.032452	0.101685	0.069674	-0.032124	0.167811	0.151890	0.277068	0.338538	0.012994	-0.017500	0.166146	0.249943	0.271224	0.159711	0.076570	0.006347	1.000000	0.206206	0.296922	0.253307
Short / medium passes per 90	0.131286	0.183557	0.140572	0.093588	0.027430	0.167663	-0.144616	-0.169798	-0.050834	-0.077697	0.118484	0.149673	-0.144371	-0.143208	0.948436	0.975833	0.735076	0.635328	0.206206	1.000000	0.388140	0.266282
Accurate passes to final third, %	0.123943	0.199179	0.073801	0.099417	0.087492	0.087917	0.056143	0.055183	0.068120	0.078119	0.110768	0.106506	0.058309	0.084530	0.359158	0.318586	0.492844	0.561493	0.296922	0.388140	1.000000	0.384419
Accurate progressive passes, %	0.137270	0.209736	0.153980	0.085371	0.123065	0.116096	0.166452	0.167727	0.234857	0.292201	0.116234	0.111926	0.152854	0.233912	0.335907	0.278130	0.239692	0.388892	0.253307	0.266282	0.384419	1.000000

Afterwards, I ended up scaling the data to make sure that they can be used along each other better, which often results into higher performances:

```python
# Calculate Standardization of Minutes and Matches played
mid_def[["Minutes played", "Matches played"]] = scaler.fit_transform(mid_def[["Matches played", "Minutes played"]])

# Calculate the robust scale of Goals, xG and Non-penalty goal
mid_def[["Goals", "xG", "Non-penalty goals"]] = robust.fit_transform(mid_def[["Goals", "xG", "Non-penalty goals"]])
# Calculate robust scale of Assists and xA
mid_def[["Assists", "xA"]] = robust.fit_transform(mid_def[["Assists", "xA"]])

# Calculate MinMax scale for Passes per 90, Received passes per 90 and Short / medium passes per 90
mid_def[["Passes per 90", "Received passes per 90", "Short / medium passes per 90"]] = min_max.fit_transform(mid_def[["Passes per 90", "Received passes
per 90", "Short / medium passes per 90"]])

# Calculate Max Abs Scale for Accurate passes, % and Accurate forward passes, %
mid_def[["Accurate passes, %", "Accurate forward passes, %"]] = abs_scaler.fit_transform(mid_def[["Accurate passes, %", "Accurate forward passes, %"]])

# Save the X and y values
mid_def_X = mid_def.drop(["Market value", "Bin"], axis=1)
mid_def_y = mid_def["Bin"]
# Save the y value for Linear Regression.
mid_def_y_lin = mid_def["Market value"]
```

Python

And afterwards, I applied parameter selection for certain models, below I will only show one example, since I did this using 2 methods for 2 models for each position.

Goalkeeper, GridSearch

```python
# Define grid of parameters
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [2, 4, 6, None],
    'min_samples_split': [2, 4, 6]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5, n_jobs=-1)

# Train GridSearchCV
grid_search.fit(gk_X, gk_y)

# Print out best parameters and scores
print("Best parameters found: ", grid_search.best_params_)
print("Best score found: ", grid_search.best_score_)
```

Python

```
Best parameters found:  {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best score found:  0.56796875
```

After this, I trained the new models with updated data and better parameters, which ended up with the following performances: (Note: These performances were for goalkeepers, I had a different model for every type of position.)

Linear Regression

```
# Create the Linear Regression Model
lr = LinearRegression()
# Train the model
lr.fit(X_train_lin, y_train_lin)

# Predict using the test set
y_pred_lin = lr.predict(X_test_lin)

# Calculate the Linear Regression scores.
mse = mean_squared_error(y_test_lin, y_pred_lin)
rmse = mse ** 0.5
r2 = r2_score(y_test_lin, y_pred_lin)

# Print out the Linear Regression scores.
print("Score testing data:", lr.score(X_test_lin, y_test_lin))
print("Score training data:", lr.score(X_train_lin, y_train_lin))
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-Square:", r2)
print("Cross Validation Score:", cross_val_score(lr, X, y))
```

```
Score testing data: 0.19894303780572276
Score training data: 0.15032932151849643
Mean Squared Error: 839668989385.819
Root Mean Squared Error: 916334.5401030232
R-Square: 0.19894303780572276
Cross Validation Score: [0.05256356 0.04590991 0.05634342 0.02373568 0.13647904]
```

Logistic Regression

```
# Create Logistic Regression Model.
log_reg = LogisticRegression(max_iter=100000)

# Train the model
log_reg.fit(X_train, y_train)

# Make a prediction
y_pred_log = log_reg.predict(X_test)

# Calculate the Logistic Regression scores
mse = mean_squared_error(y_test, y_pred_log)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred_log)
accuracy = accuracy_score(y_test, y_pred_log)

# Print out the scores.
print("Score testing data:", log_reg.score(X_test, y_test))
print("Score training data:", log_reg.score(X_train, y_train))
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-Square:", r2)
print("Accuracy Score:", accuracy)
print(classification_report(y_test, y_pred_log))
print("Cross Validation Score:", cross_val_score(log_reg, X, y))
sns.heatmap(confusion_matrix(y_test, y_pred_log))
plt.show()
```

```
Score testing data: 0.5833333333333334
Score training data: 0.609375
Mean Squared Error: 0.5572916666666666
Root Mean Squared Error: 0.7465197027987048
R-Square: 0.10576201099080451
Accuracy Score: 0.5833333333333334
              precision    recall  f1-score   support

           0       0.64      0.70      0.67       155
           1       0.48      0.53      0.50       135
           2       0.68      0.48      0.56        94

    accuracy                           0.58       384
   macro avg       0.60      0.57      0.58       384
weighted avg       0.59      0.58      0.58       384

Cross Validation Score: [0.52056261 0.50175814 0.51949243 0.49923547 0.47568807]
```

Random Forest Classifier

```
# Create the Model
clf = RandomForestClassifier(max_depth=None, min_samples_split=2, n_estimators=91)

# Train model
clf.fit(X_train, y_train)

# Predict the result
y_pred_clf = clf.predict(X_test)

# Calculating the scores
mse = mean_squared_error(y_test, y_pred_clf)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred_clf)
accuracy = accuracy_score(y_test, y_pred_clf)

# Print out the RandomForestClassifier scores
print("Score testing data:", clf.score(X_test, y_test))
print("Score training data:", clf.score(X_train, y_train))
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-Square:", r2)
print("Accuracy Score:", accuracy)
print(classification_report(y_test, y_pred_clf))
print("Cross Validation Score:", cross_val_score(clf, X, y))
sns.heatmap(confusion_matrix(y_test, y_pred_clf))
plt.show()
```

```
Score testing data: 0.5520833333333334
Score training data: 1.0
Mean Squared Error: 0.6197916666666666
Root Mean Squared Error: 0.7872684844871326
R-Square: 0.005473638391642388
Accuracy Score: 0.5520833333333334
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       155
           1       0.45      0.53      0.49       135
           2       0.61      0.41      0.49        94

    accuracy                           0.55       384
   macro avg       0.56      0.53      0.54       384
weighted avg       0.56      0.55      0.55       384

Cross Validation Score: [0.50496866 0.49441981 0.53095857 0.49082569 0.45993884]
```

Gradient Boosting Trees

```
# Initiate the gradient booster classifier
gb = XGBClassifier(max_depth=2, n_estimators=105)

# Train the model.
gb.fit(X_train, y_train)

# Predict the values
y_pred_gb = gb.predict(X_test)

# Calculate scores.
mse = mean_squared_error(y_test, y_pred_gb)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred_gb)
accuracy = accuracy_score(y_test, y_pred_gb)

# Print out the performances of the model.
print("Score testing data:", gb.score(X_test, y_test))
print("Score training data:", gb.score(X_train, y_train))
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-Square:", r2)
print("Accuracy Score:", accuracy)
print(classification_report(y_test, y_pred_gb))
print("Cross Validation Score:", cross_val_score(gb, X, y))
sns.heatmap(confusion_matrix(y_test, y_pred_gb))
plt.show()
```

```
Score testing data: 0.5703125
Score training data: 0.8314732142857143
Mean Squared Error: 0.609375
Root Mean Squared Error: 0.7806247497997998
R-Square: 0.02218836715816941
Accuracy Score: 0.5703125
              precision    recall  f1-score   support

           0       0.62      0.63      0.63       155
           1       0.48      0.53      0.50       135
           2       0.65      0.52      0.58        94

    accuracy                           0.57       384
   macro avg       0.58      0.56      0.57       384
weighted avg       0.58      0.57      0.57       384

Cross Validation Score: [0.54104877 0.52560771 0.55373796 0.48883792 0.47859327]
```

These are the final performances for goalkeepers. We print out multiple types of scores, so we can decide which model is best fitted for the question based on multiple factors and performances.