Springboard Capstone 2 Milestone Report 2
Aaron Kochman

Scouting Potential Premier League Signings with Machine Learning

Introduction

Problem Statement

Elite soccer teams around the world are constantly on the lookout for players to give their squad the advantage they need to win more games, tournaments, and ultimately increase profits. Machine learning can be utilized with data surrounding player value, game stats, and compiled metrics of players' abilities based on previous seasons in video games like FIFA 20 as well as online data repositories.

Building a scouting report for a premier league team based on player attributes and value could be a valuable tool for scouting replacement players or potential signings that can increase a club's performance.

Business Case

Data Scientists employed by organizations like a top-tiered English Premier League Football Club could be asked to manage and recommend transfer targets. In addition to recommending players for the squad, they also may be asked to predict transfer values or develop metrics to score player's skills or characteristics.

This project utilizes web scraping and machine learning to analyze player performance, transfer value, and game statistics to predict signings and potential values for my favorite Premier League club Arsenal.

Machine Learning Utilized

Clustering

- Scikit-learn k-means Clustering
    - Utilized for clustering player positions and initial analysis

Dimensionality Reduction

- Principal Component Analysis

Nearest Neighbors

- Scikit-learn neighbors KDtree
    - Utilized to determine nearest neighbors for Arsenal players

Models for Transfer Value Predictions based on PCA

- XGBoost Linear Regression
- Scikit-learn Linear Regression

Hypotheses

Null Hypothesis (H0): There is no relationship between player values and skill level.

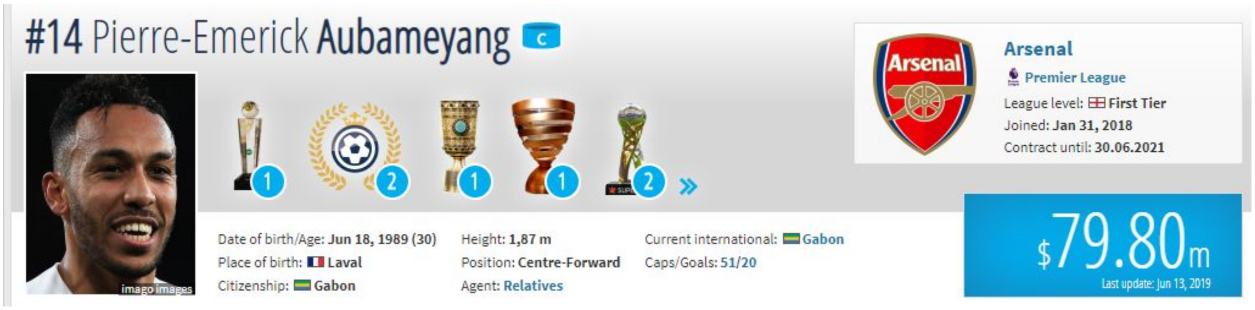
Alternative Hypothesis (Ha): There is some relationship between player values and skill level.

Data Sources

In order to explore different datasets to build prediction models, different techniques were used to wrangle, clean, and combine datasets from selected sources. The three main datasets produced in this project stemmed from SoFIFA, transfermarkt, and Fantasy Premier League.

transfermarkt

Transfermarkt tracks and compiles transfer data from multiple soccer leagues globally as well as compiles statistics including goals, assists, and player accolades.



#14 Pierre-Emerick Aubameyang

Arsenal
Premier League
League level: First Tier
Joined: Jan 31, 2018
Contract until: 30.06.2021

Date of birth/Age: Jun 18, 1989 (30)
Place of birth: Laval
Citizenship: Gabon

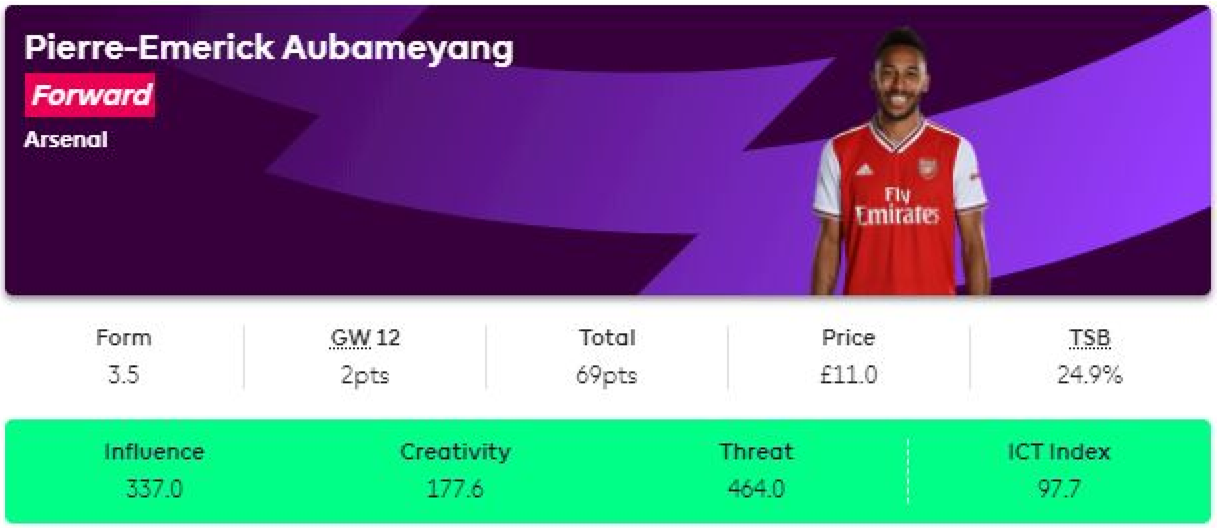
Height: 1,87 m
Position: Centre-Forward
Agent: Relatives

Current international: Gabon
Caps/Goals: 51/20

$79.80m
Last update: Jun 13, 2019

Fantasy Premier League

Fantasy Premier League contains attributes as well as real-world statistics. The main component on FPL that is useful for determining a player's value are components like Influence, Creativity, and Threat.



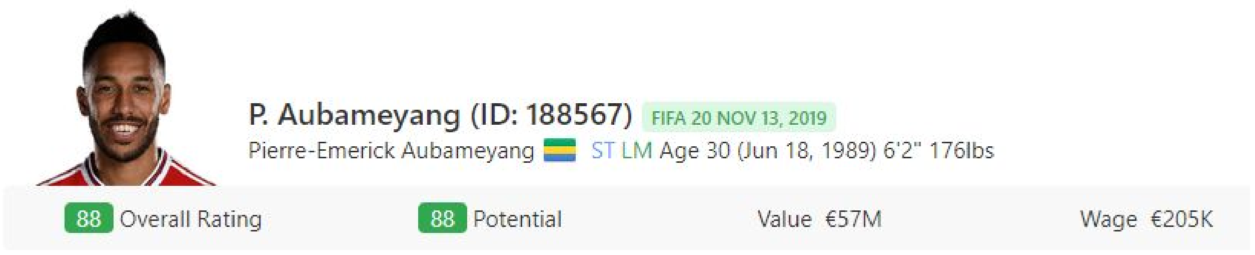
Pierre-Emerick Aubameyang
Forward
Arsenal

Form	GW 12	Total	Price	TSB
3.5	2pts	69pts	£11.0	24.9%

Influence	Creativity	Threat	ICT Index
337.0	177.6	464.0	97.7

SoFIFA

SoFIFA compiles player data from the video game franchise FIFA, produced by EA Sports. Player data on sofifa includes statistics like Speed, Agility, Aggression, Ball Control, Shooting, as well as transfer values.



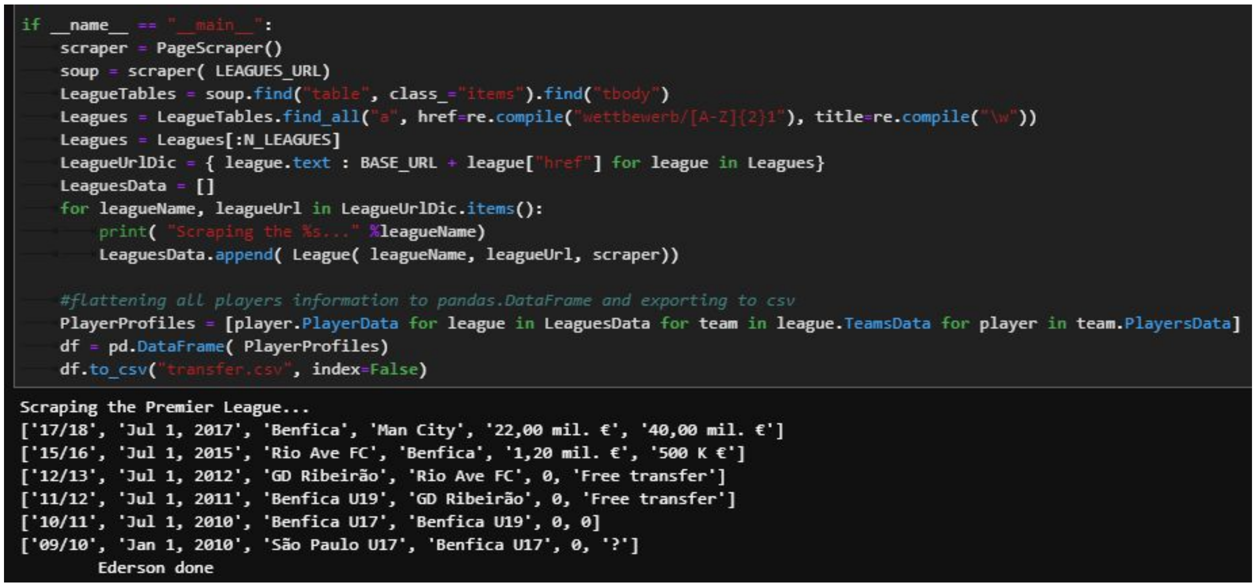
P. Aubameyang (ID: 188567) FIFA 20 NOV 13, 2019
Pierre-Emerick Aubameyang ST LM Age 30 (Jun 18, 1989) 6'2" 176lbs

88 Overall Rating   88 Potential   Value €57M   Wage €205K

Data Wrangling

transfermarkt Data Wrangling

Transfermarkt is HTML based thus the Python package Beautiful Soup was used for parsing out transfer data as well as player statistics. In essence each of the players profiles contained a unique URL and each statistic or metric are encased in HTML classes including ‘responsive-table’, ‘td’, and ‘tbody’.



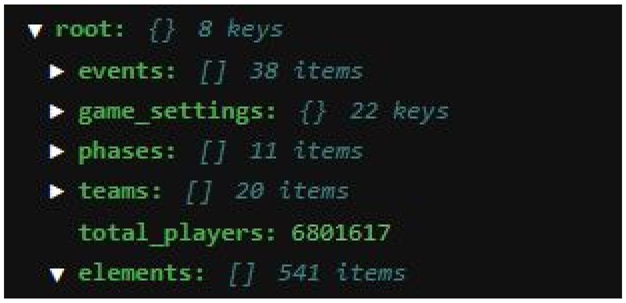
```
if __name__ == "__main__":
    scraper = PageScraper()
    soup = scraper( LEAGUES_URL)
    LeagueTables = soup.find("table", class_="items").find("tbody")
    Leagues = LeagueTables.find_all("a", href=re.compile("wettbewerb/[A-Z]{2}1"), title=re.compile("\w"))
    Leagues = Leagues[:N_LEAGUES]
    LeagueUrlDic = { league.text : BASE_URL + league["href"] for league in Leagues}
    LeaguesData = []
    for leagueName, leagueUrl in LeagueUrlDic.items():
        print( "Scraping the %s..." %leagueName)
        LeaguesData.append( League( leagueName, leagueUrl, scraper))

    #flattening all players information to pandas.DataFrame and exporting to csv
    PlayerProfiles = [player.PlayerData for league in LeaguesData for team in league.TeamsData for player in team.PlayersData]
    df = pd.DataFrame( PlayerProfiles)
    df.to_csv("transfer.csv", index=False)
```

```
Scraping the Premier League...
['17/18', 'Jul 1, 2017', 'Benfica', 'Man City', '22,00 mil. €', '40,00 mil. €']
['15/16', 'Jul 1, 2015', 'Rio Ave FC', 'Benfica', '1,20 mil. €', '500 K €']
['12/13', 'Jul 1, 2012', 'GD Ribeirão', 'Rio Ave FC', 0, 'Free transfer']
['11/12', 'Jul 1, 2011', 'Benfica U19', 'GD Ribeirão', 0, 'Free transfer']
['10/11', 'Jul 1, 2010', 'Benfica U17', 'Benfica U19', 0, 0]
['09/10', 'Jan 1, 2010', 'São Paulo U17', 'Benfica U17', 0, '?']
        Ederson done
```

After scraping transfermarkt for performance statistics (goals, assists, games played, etc.) as well as transfer valuation, each data set was parsed out by season (17/18, 18/19, 19/20, etc.) and cleaned to produce formatting that would be standardized between datasets. Data from each scrape was joined based on player name and season year, but not every player had transfer data for certain years.

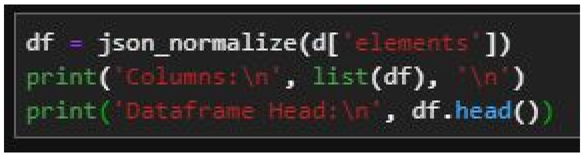
Fantasy Premier League Data Wrangling

Fantasy Premier League contains an API that can be called using a JSON request. Once connected to the API, a JSON file was downloaded containing the relevant data.



```
▼ root: {} 8 keys
  ▶ events: [] 38 items
  ▶ game_settings: {} 22 keys
  ▶ phases: [] 11 items
  ▶ teams: [] 20 items
    total_players: 6801617
  ▼ elements: [] 541 items
```

The JSON file was then normalized with the json_normalize package from pandas.io.json and exported as a CSV.



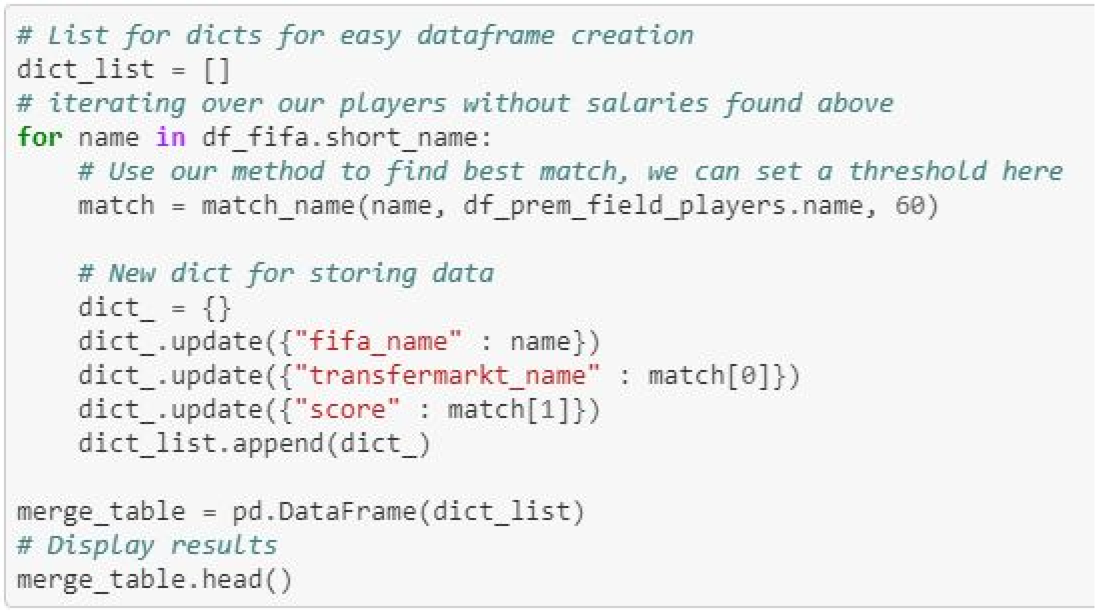
```
df = json_normalize(d['elements'])
print('Columns:\n', list(df), '\n')
print('Dataframe Head:\n', df.head())
```

SoFIFA (Kaggle) Data Acquisition

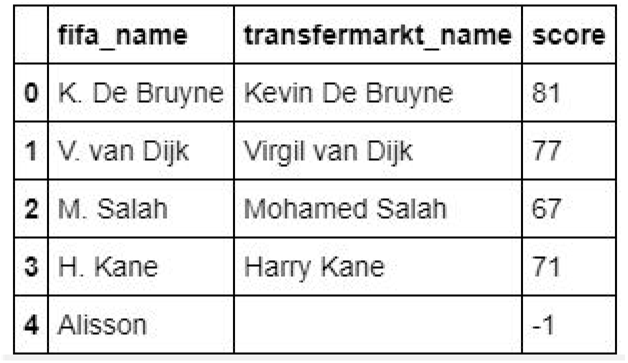
A useful dataset on Kaggle produced by user stefanoleone992 was made available near the beginning of this project. They had scraped SoFIFA and compiled a CSV of every player from FIFA 15-20 (6 FIFA video games and 15,458 players throughout multiple years). This dataset was used heavily during this project and many thanks goes out to the Kaggle user who compiled the data.

Data Compiling and Matching Player Names with FuzzyWuzzy

After each data source was cleaned and processed, I was ready to combine datasets. The major problem during this step was matching names that were not formatted in a standard method to explore holistically.



```
# List for dicts for easy dataframe creation
dict_list = []
# iterating over our players without salaries found above
for name in df_fifa.short_name:
    # Use our method to find best match, we can set a threshold here
    match = match_name(name, df_prem_field_players.name, 60)

    # New dict for storing data
    dict_ = {}
    dict_.update({"fifa_name" : name})
    dict_.update({"transfermarkt_name" : match[0]})
    dict_.update({"score" : match[1]})
    dict_list.append(dict_)

merge_table = pd.DataFrame(dict_list)
# Display results
merge_table.head()
```



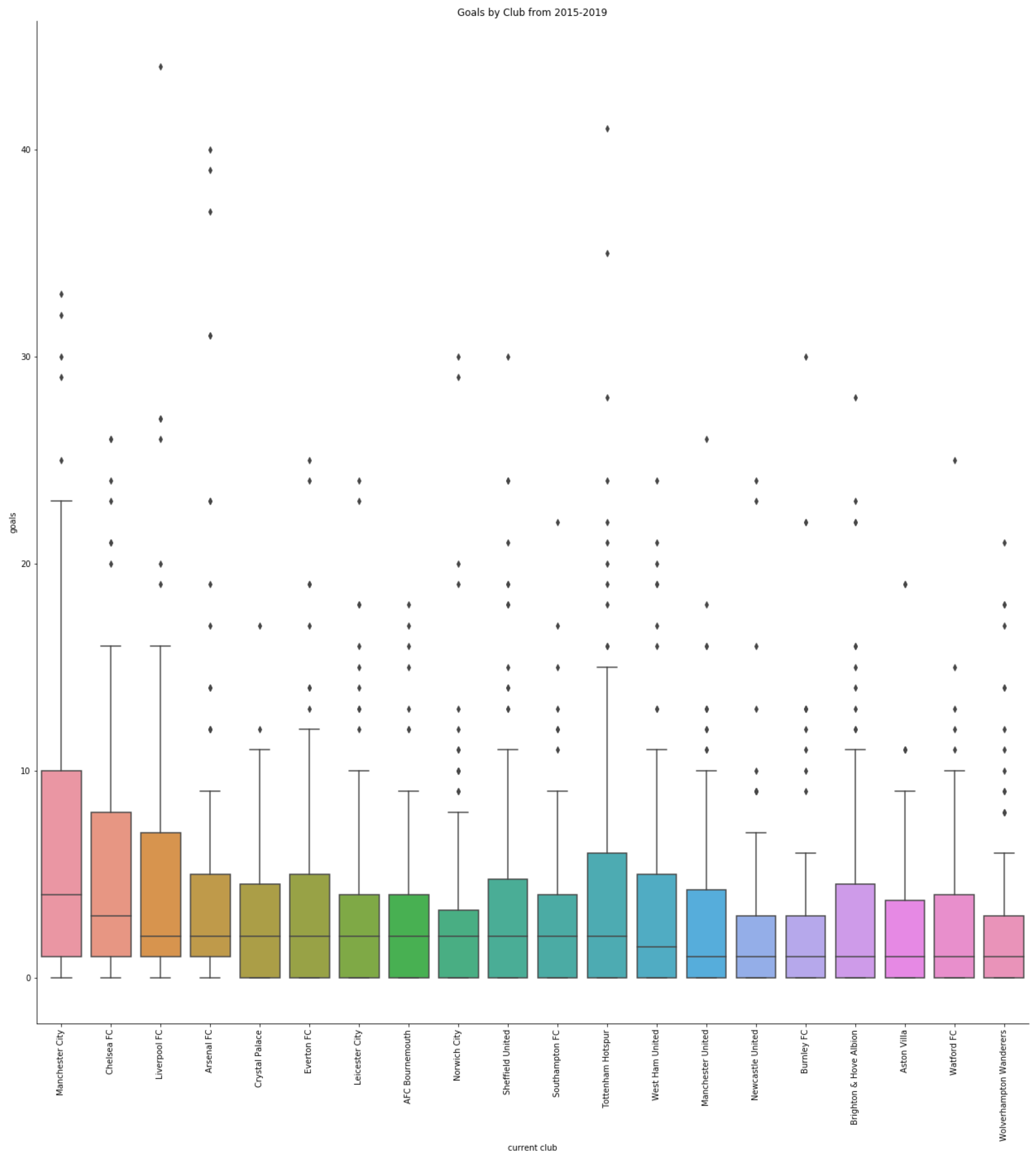
	fifa_name	transfermarkt_name	score
0	K. De Bruyne	Kevin De Bruyne	81
1	V. van Dijk	Virgil van Dijk	77
2	M. Salah	Mohamed Salah	67
3	H. Kane	Harry Kane	71
4	Alisson		-1

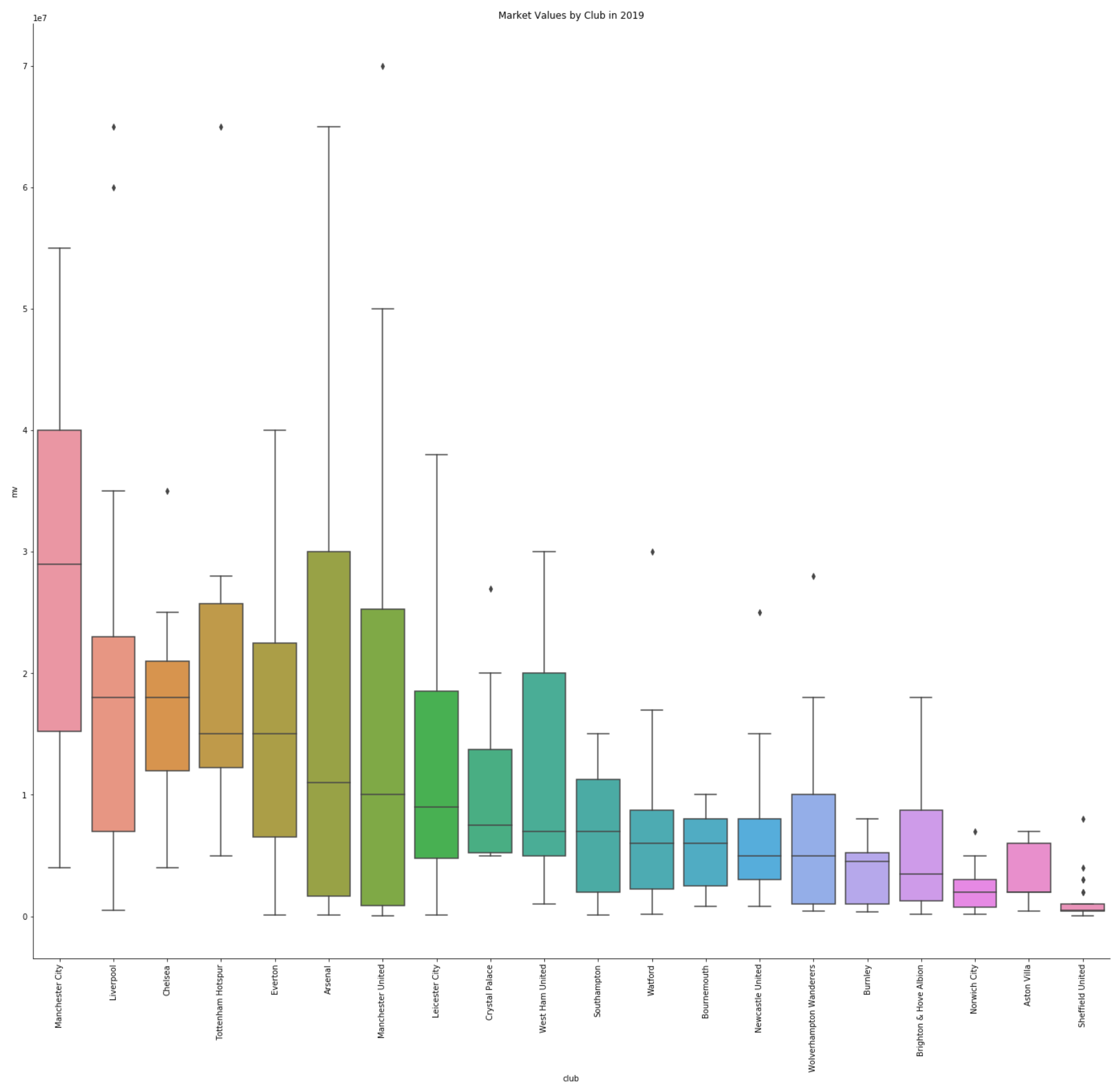
Once each player was fuzzy matched between datasets, the matching data was joined together and data that did not match was dropped. This process was repeated to join transfermarkt data and FPL data to the FIFA dataset.

The resulting data was analyzed and explored to determine use but consequently, it was determined that too much data was being lost in the fuzzy matching phase to provide enough training data to use during machine learning.

Exploratory Analysis and Exploratory Machine Learning

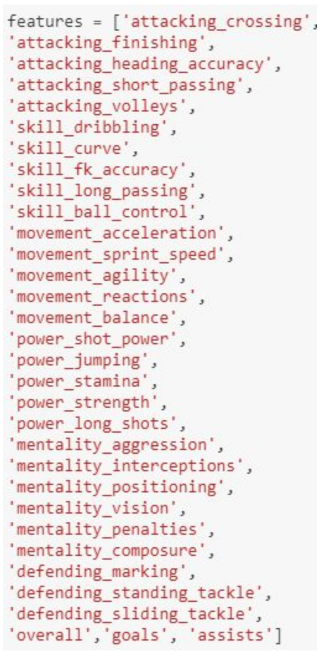
From the combined data and individual data sources, a variety of questions were initially asked including, what is the team in the English Premier League with the highest market value (Manchester City), which team typically scores the most goals (Manchester City), and how could the data be prepared to create an accurate machine learning model.





It can be observed from the data that higher valued teams tend to score more goals and have larger budgets for players. It can also be determined that players who are currently on higher valued teams contain higher valued players who score more goals in the league.

Principal Component Analysis (PCA)

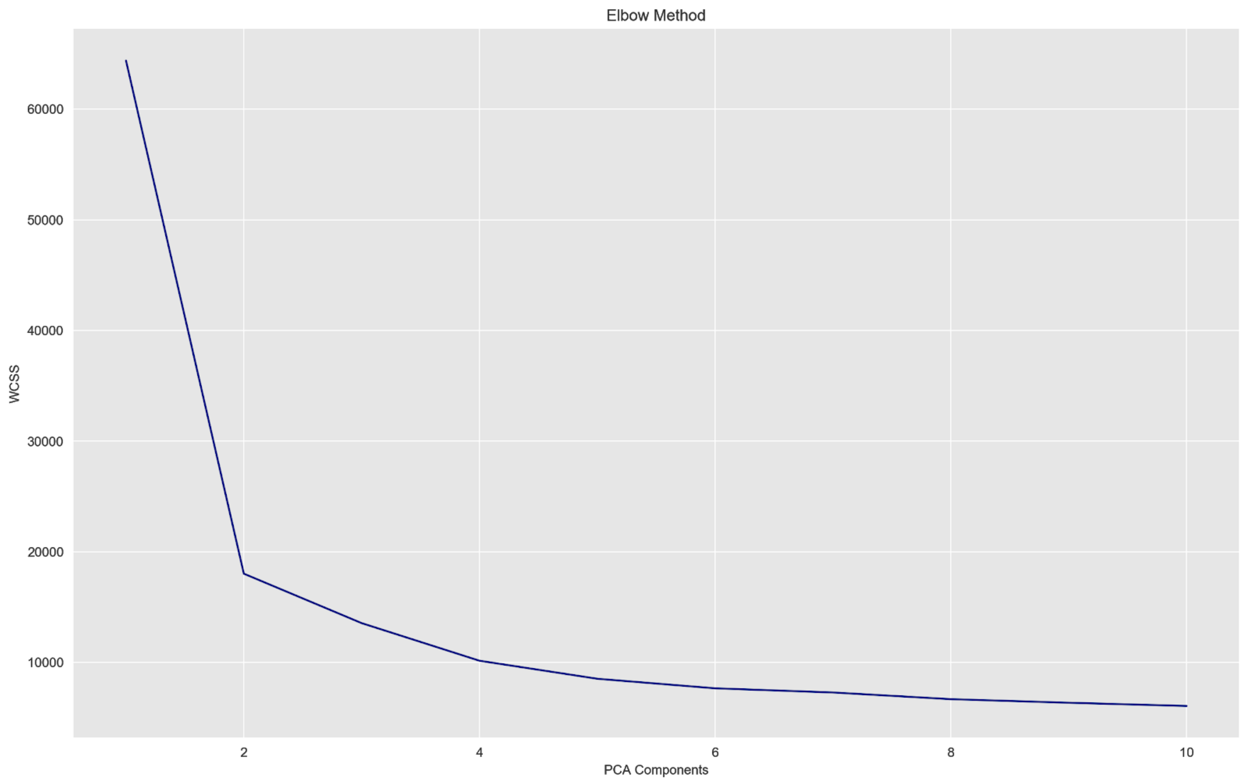


```
features = ['attacking_crossing',
'attacking_finishing',
'attacking_heading_accuracy',
'attacking_short_passing',
'attacking_volleys',
'skill_dribbling',
'skill_curve',
'skill_fk_accuracy',
'skill_long_passing',
'skill_ball_control',
'movement_acceleration',
'movement_sprint_speed',
'movement_agility',
'movement_reactions',
'movement_balance',
'power_shot_power',
'power_jumping',
'power_stamina',
'power_strength',
'power_long_shots',
'mentality_aggression',
'mentality_interceptions',
'mentality_positioning',
'mentality_vision',
'mentality_penalties',
'mentality_composure',
'defending_marking',
'defending_standing_tackle',
'defending_sliding_tackle',
'overall','goals', 'assists']
```

After it was ensured the data was comparable to real world assumptions, the combined data was split into a 2-Component Principal Component Analysis based on features that stemmed mostly from the FIFA data set. The logic behind the PCA analysis is to consolidate a many dimensional data set into as few dimensions as possible, while still maintaining the integrity of the original data. PCA is also a valuable tool for visualizing a variety of features compiled together.

An elbow plot was used to depict the correct number of PCA components to use for our data. The elbow plot shows that PCA variance of the combined data that two components would be sufficient to analyze. As the features are condensed into PCA values, the idea is the new condensed values will generally depict the same data story as the original features, but in a much more useable format.

The 2-Component PCA was validated using an explained variance ratio, in which the two PCA components covered over 85% of the data.



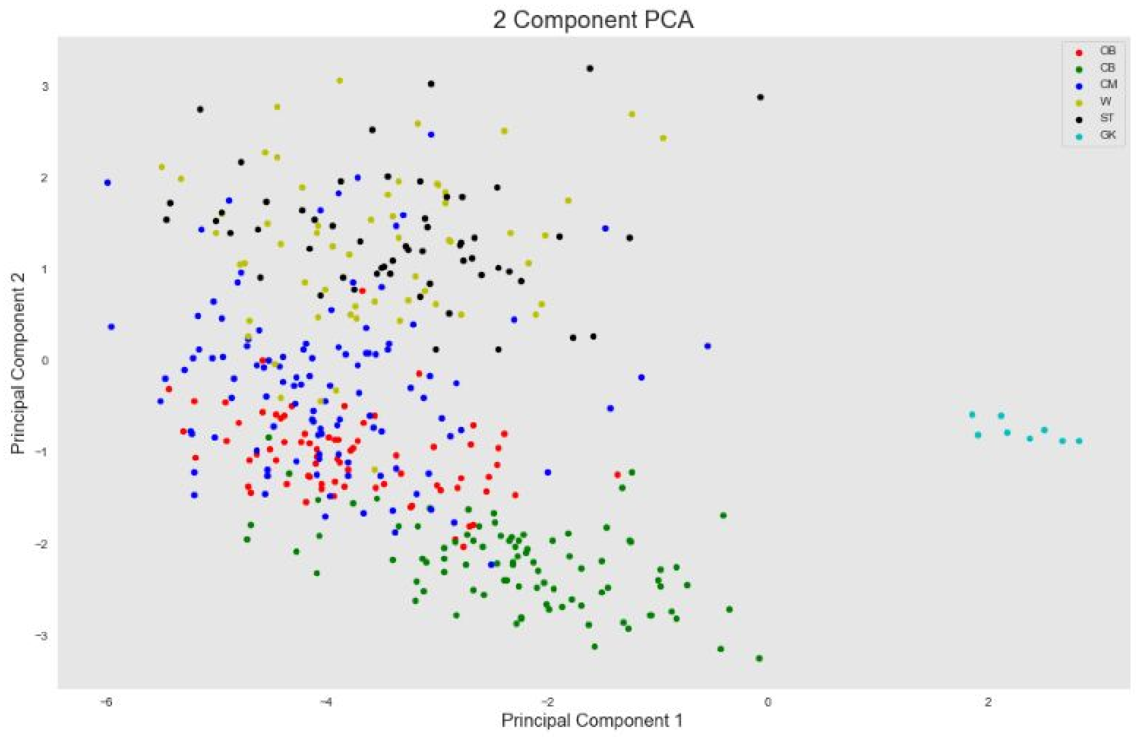
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head()
```

	principal component 1	principal component 2
0	-3.270765	-2.164194
1	-3.517926	-1.864947
2	-3.411034	-1.868319
3	-3.663187	-2.139183
4	-3.400502	-2.167475

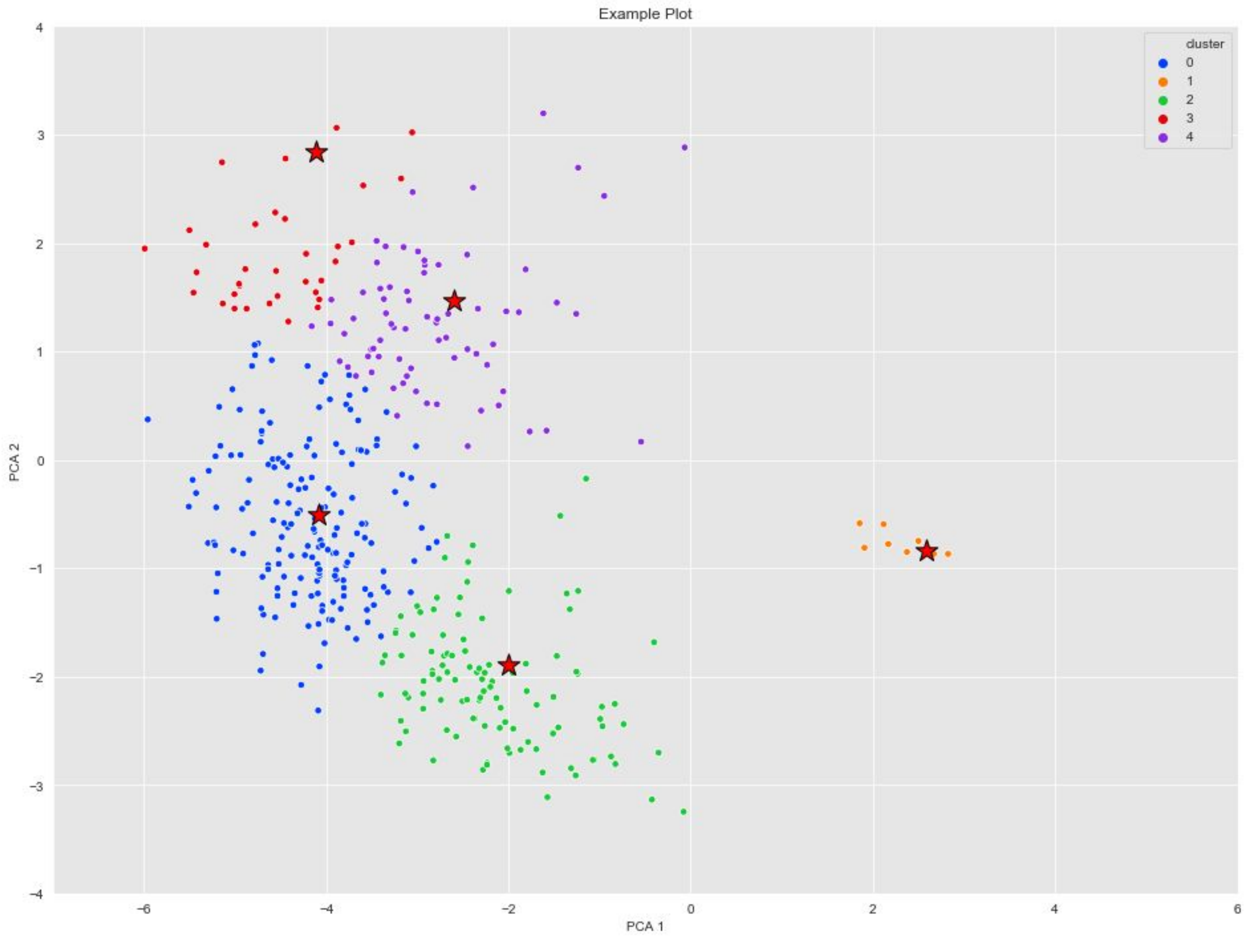
```python
pca.explained_variance_ratio_
```

```
array([0.80055365, 0.08004547])
```

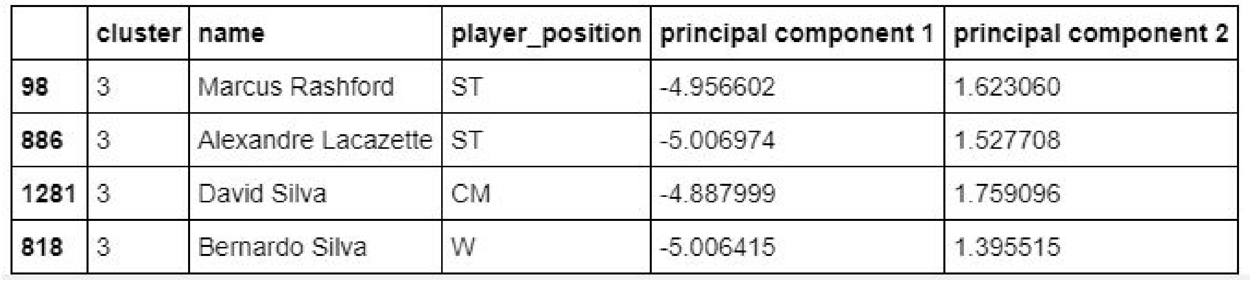
The first trial of the 2-Component PCA on the combined dataset showed that ultimately, goalkeepers were outliers as expected. It was observed that based on the PCA, defenders are fairly different from midfielders or strikers, but midfielders seem to overlap with strikers quite frequently.



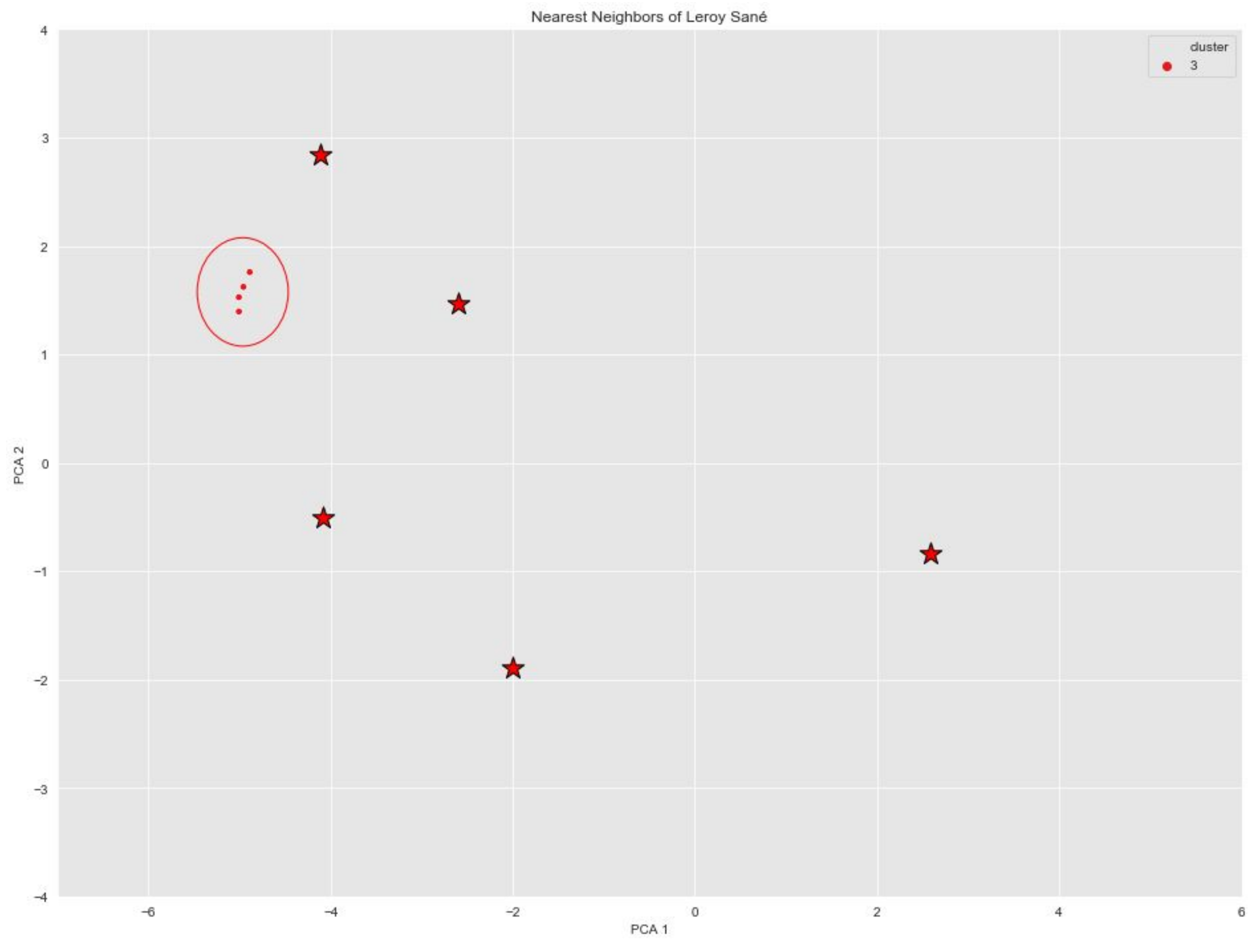
K-means clustering was used to establish clusters for each player based on position to verify that nearest neighbors could be extrapolated accurately. Centroids for each position group could be utilized to predict player positions based on the PCA components.



Nearest neighbors were determined for a striker, Leroy Sané, and the idea of producing a scouting report followed. By using euclidean distance, scikit-learn’s KDTree can work from a root point and measures the nearest neighbors of that root point in a tree format. The nearest neighbors method used can be visualized to confirm neighbor players to Sané were indeed near each other when plotted against each other.



	cluster	name	player_position	principal component 1	principal component 2
98	3	Marcus Rashford	ST	-4.956602	1.623060
886	3	Alexandre Lacazette	ST	-5.006974	1.527708
1281	3	David Silva	CM	-4.887999	1.759096
818	3	Bernardo Silva	W	-5.006415	1.395515

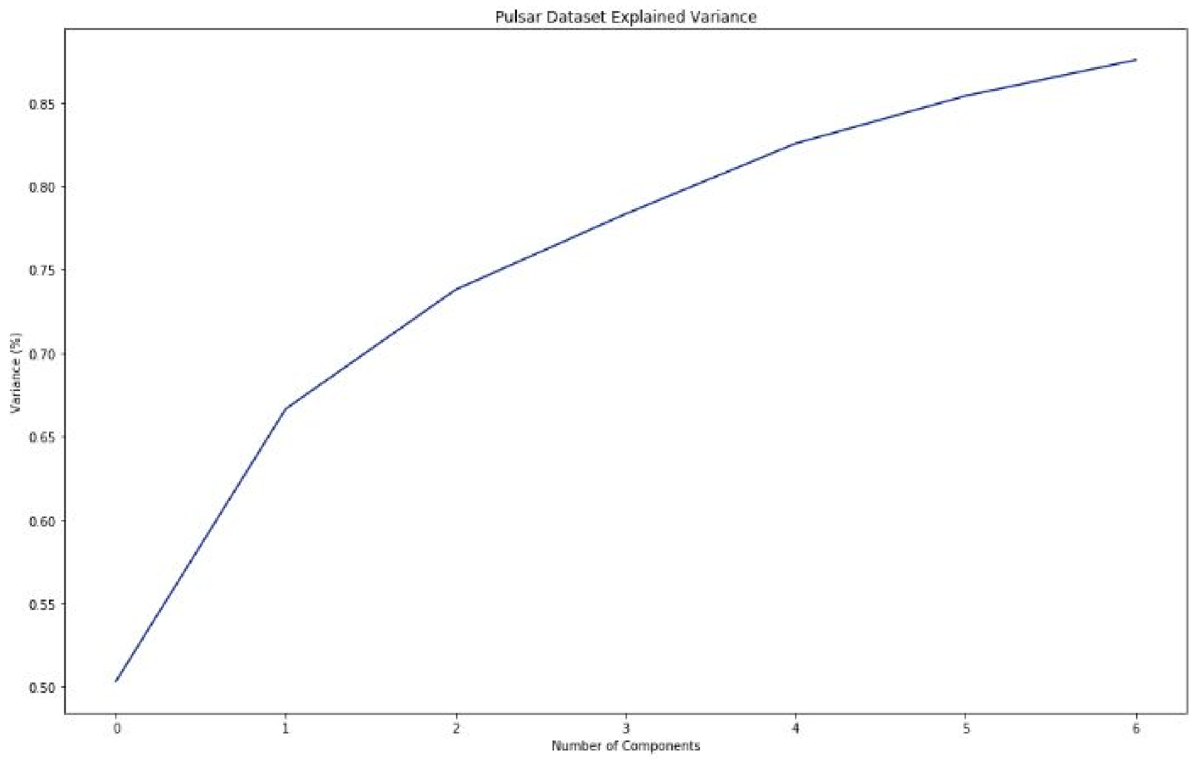


Once it was determined that the data could be used to cluster players together, it was decided to run machine learning on the entire FIFA dataset, excluding the FPLand transfermarkt datasets so that more players in a consistent data format would be combined together. Much more cleaning, transforming, and processing would be required if the transfermarkt and FPL datasets were to be utilized.

Transformations and Data Preparation for Machine Learning

PCA Data Set

As mentioned previously, player features from the FIFA dataset were compiled into PCA components. 6 PCA components were required for the entire FIFA dataset compared to the subset of combined data previously analyzed.



Compiling Current Arsenal Players

To begin the process of predicting potential replacement players for the entire Arsenal squad, the 18/19 and 19/20 season Arsenal players were extracted from the dataset.

```
df_arsenal = df.query('club == "Arsenal" & year == "18/19"')
```

Once the Arsenal players were parsed out from the dataset, the next task was to obtain a list of players that contained similar PCA values to the 33 Arsenal players.

The potential transfer players were then paired with the Arsenal player that they were neighbors with. It turns out that some of the Arsenal players were neighbors with each other, so those players were dropped out of the final scouting report.

```
#Importing KDTree

from sklearn.neighbors import KDTree

kdt = KDTree(df[['pc1', 'pc2', 'pc3', 'pc4', 'pc5', 'pc6']])
```

```
#Using KDTree to find 4 players similar to that of Arsenal Players

dist, idx = kdt.query(df_arsenal_pca, k=5)
```

```
idx = idx.flatten()
```

```
idx
```

```
array([14820, 14889, 15003, 14915, 14908, 14830, 14974, 14859, 14993,
       15204, 14878, 14883, 14897, 14943, 14940, 14889, 15111, 14915,
       14820, 14912, 14898, 14831, 15025, 14843, 14980, 14907, 14903,
       15008, 14913, 15041, 14991, 15135, 15157, 15352, 15700, 14997,
       15527, 15024, 15115, 15239, 15013, 14944, 14909, 15697, 15138,
       15019, 14884, 15083,  9858, 15113, 15078,  9962, 15587, 15995,
       16075, 15148, 17032,    44, 15136, 15470, 15171,     5, 15058,
       16143, 15540, 15201, 15404, 15233, 15600, 15210, 15228, 15201,
       15404,  6407, 15600, 15431, 15285, 15297, 15882, 15445, 15467,
       15613, 16096, 15364, 17551, 15825, 14936, 15748, 16636,  6408,
       15900, 17200, 17675, 10291, 10263, 16711, 10667, 17384, 17540,
       17508, 17816, 17862, 22010, 20312, 23475, 18347, 22452,   244,
        6940, 21828, 20274, 10521, 22576, 19323, 19441, 20763, 25431,
       21302, 11613, 23731, 20847, 19945, 20999, 22850, 21927, 21742,
       22575, 20018, 21115, 19622, 23848, 12554, 25731, 21845, 12453,
       24700,   947,  1004, 24101, 12073, 25893, 26094, 26864, 28034,
        8589, 27417, 12956, 25909, 26289, 13652, 27438,  2006, 13037,
       26876, 26913, 28119, 27667, 28288, 27858,  1863, 29246,  2150,
       30674, 29889,  2653], dtype=int64)
```

	pc1	pc2	pc3	pc4	pc5	pc6	value_eur	short_name	club	transfer
67812	-6.90	-2.46	2.86	-1.86	1.56	-0.24	50500000	P. Aubameyang	Arsenal	P. Aubameyang
67890	-6.70	-2.19	2.54	-1.34	1.36	-0.42	36500000	A. Lacazette	Arsenal	P. Aubameyang
68017	-6.17	-2.19	2.40	-1.98	1.13	-0.36	26000000	C. Bakambu	Beijing Sinobo Guoan FC	P. Aubameyang
67923	-7.08	-2.47	2.41	-1.07	1.15	0.02	35000000	M. Depay	Olympique Lyonnais	P. Aubameyang
67916	-6.77	-2.35	1.78	-1.93	1.88	-0.39	41000000	Gabriel Jesus	Manchester City	P. Aubameyang
67825	-5.92	-3.68	2.94	2.00	2.28	0.14	43500000	M. Özil	Arsenal	M. Özil
67986	-5.94	-3.28	2.47	1.70	2.09	0.28	30000000	Luis Alberto	Lazio	M. Özil
67856	-6.26	-3.93	1.93	2.39	2.48	-0.22	17000000	F. Ribéry	FC Bayern München	M. Özil
68007	-5.68	-2.75	1.91	1.37	2.23	0.05	27000000	E. Forsberg	RB Leipzig	M. Özil
68241	-5.95	-3.26	1.70	2.05	1.46	-0.24	13000000	Nani	Sporting CP	M. Özil
67878	6.99	0.12	2.62	-1.18	4.47	0.05	27000000	B. Leno	Arsenal	B. Leno
67884	6.80	-0.24	2.48	-1.22	4.63	0.11	26000000	W. Szczęsny	Juventus	B. Leno
67902	7.10	-0.28	2.80	-1.27	4.32	0.02	19000000	S. Ruffier	AS Saint-Étienne	B. Leno
67952	7.33	-0.27	2.78	-0.93	4.55	0.14	13000000	S. Mandanda	Olympique de Marseille	B. Leno
67949	6.76	-0.29	2.31	-1.61	4.66	0.12	6000000	Pepe Reina	Milan	B. Leno

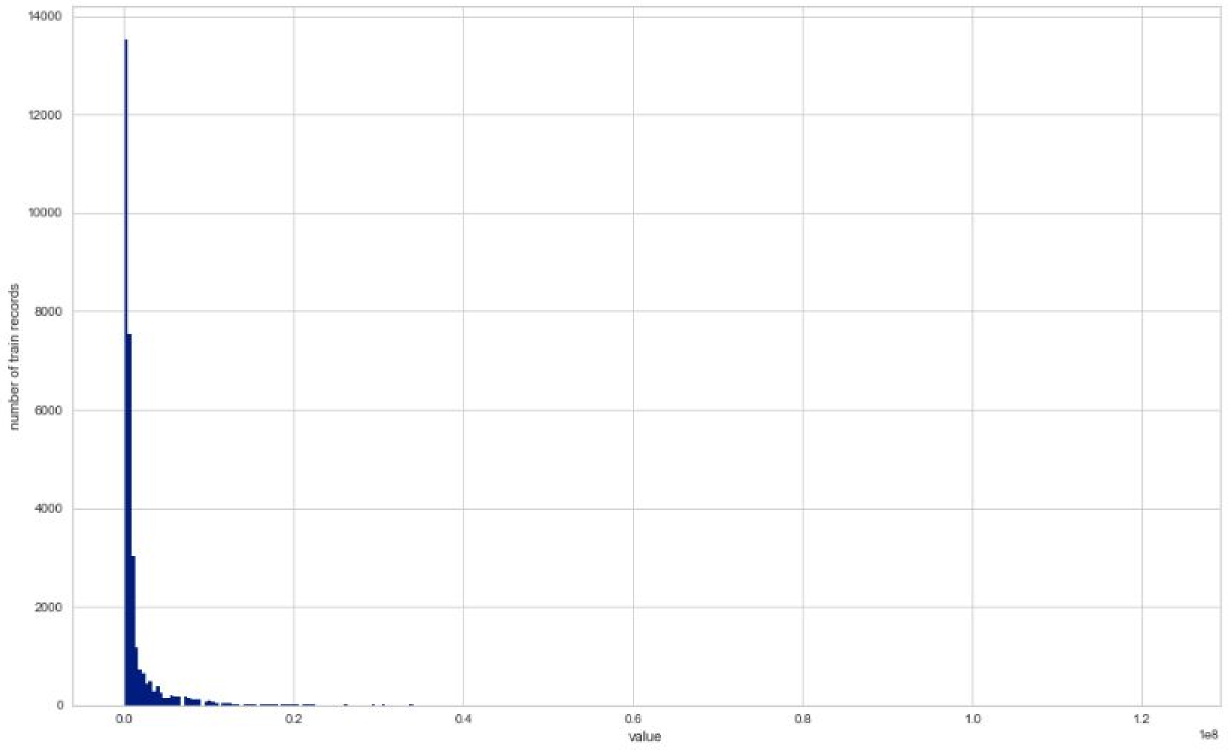
```
nn_df = nn_df.loc[nn_df['club']!='Arsenal']
nn_df
```

	pc1	pc2	pc3	pc4	pc5	pc6	value_eur	short_name	club	transfer
68017	-6.17	-2.19	2.40	-1.98	1.13	-0.36	26000000	C. Bakambu	Beijing Sinobo Guoan FC	P. Aubameyang
67923	-7.08	-2.47	2.41	-1.07	1.15	0.02	35000000	M. Depay	Olympique Lyonnais	P. Aubameyang
67916	-6.77	-2.35	1.78	-1.93	1.88	-0.39	41000000	Gabriel Jesus	Manchester City	P. Aubameyang
67986	-5.94	-3.28	2.47	1.70	2.09	0.28	30000000	Luis Alberto	Lazio	M. Özil
67856	-6.26	-3.93	1.93	2.39	2.48	-0.22	17000000	F. Ribéry	FC Bayern München	M. Özil
68007	-5.68	-2.75	1.91	1.37	2.23	0.05	27000000	E. Forsberg	RB Leipzig	M. Özil
68241	-5.95	-3.26	1.70	2.05	1.46	-0.24	13000000	Nani	Sporting CP	M. Özil
67884	6.80	-0.24	2.48	-1.22	4.63	0.11	26000000	W. Szczęsny	Juventus	B. Leno
67902	7.10	-0.28	2.80	-1.27	4.32	0.02	19000000	S. Ruffier	AS Saint-Étienne	B. Leno
67952	7.33	-0.27	2.78	-0.93	4.55	0.14	13000000	S. Mandanda	Olympique de Marseille	B. Leno
67949	6.76	-0.29	2.31	-1.61	4.66	0.12	6000000	Pepe Reina	Milan	B. Leno

Training Data Processing and Transformation

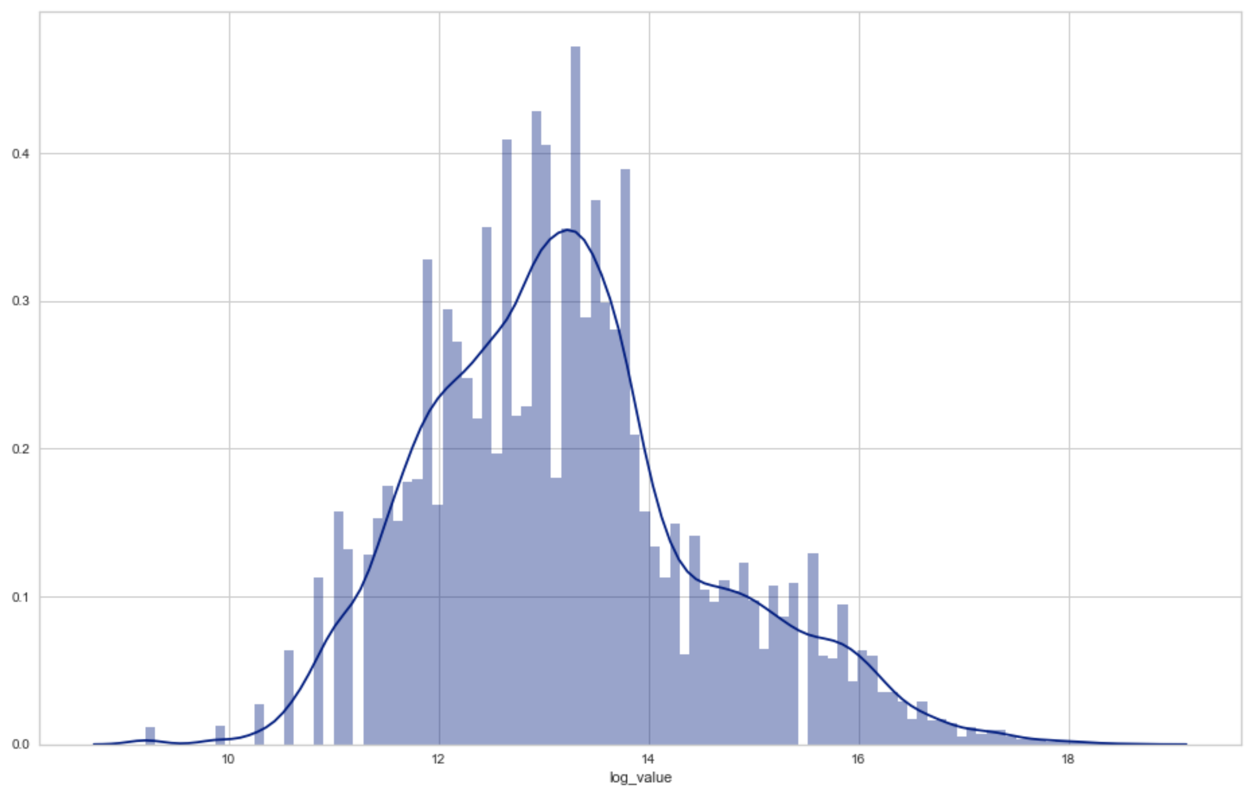
```
df_pca = df[['pc1', 'pc2', 'pc3', 'pc4', 'pc5', 'pc6', 'value_eur', 'short_name','club']]
df_pca = df_pca[df_pca.value_eur != 0]
```

```
plt.rcParams['figure.figsize'] = [16, 10]
plt.hist(df_pca['value_eur'], bins=300)
plt.xlabel('value')
plt.ylabel('number of train records')
plt.show()
```

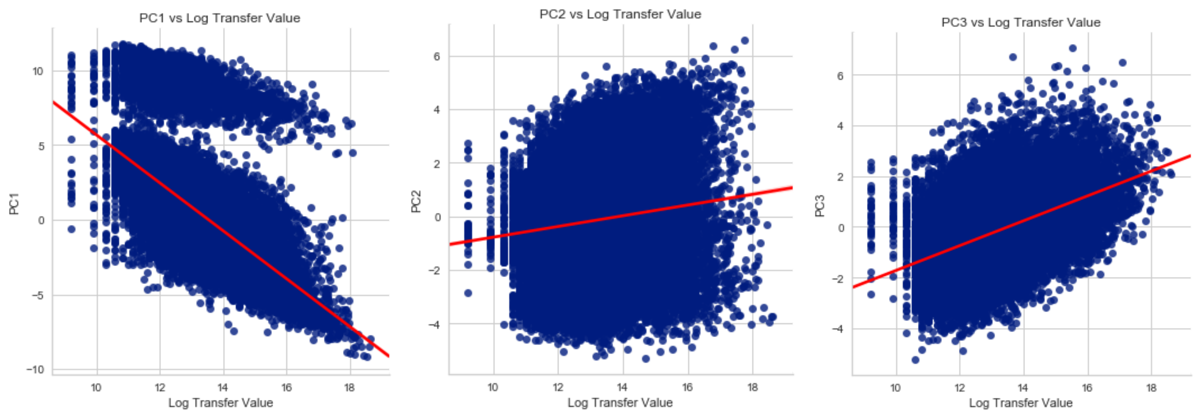


Since the player value column was the most important column for machine learning, it needed to be normalized. Zero values were dropped and even still there was an enormous skew to the data, which makes sense because there are a very few select players like Messi, Neymar, and Ronaldo who are worth well over 90 million euro, while most players fall in the range of under 100,000 euro.

Due to this nature of the market, taking the natural logarithm of the player values transformed the data into a beautiful transformation that followed a rough normal distribution.



PCA Components and Existing Player Values



When plotted against the log values, the PCA components reflected a very rough correlation. Individual PCA components would not be sufficient to predict accurate values for players so all six needed to be used in to produce the most accurate results.

Machine Learning

Nearest Neighbors Selection

```
selection = nn_df.index.to_list()
selection
```

In order to make sure the nearest neighbors were accounted for, a separate selection list was created of the neighbor indices.

scikit-learn Linear Regression

Once the data was transformed and useable for machine learning, the data was split into training and test sets with the neighbor players appended to the test set to make sure predictions applied to them.

```
from sklearn.model_selection import train_test_split


X = df_pca.drop(['log_value','value_eur', 'club', 'short_name'], axis=1)
y = df_pca[['log_value']]
```

```
X_selected = X.loc[selection]
y_selected = y.loc[selection]


# Split X and y into X_
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
```

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
pred = regressor.predict(X_test)
print('Linear Regression R squared": %.4f' % regressor.score(X_test, y_test))
```

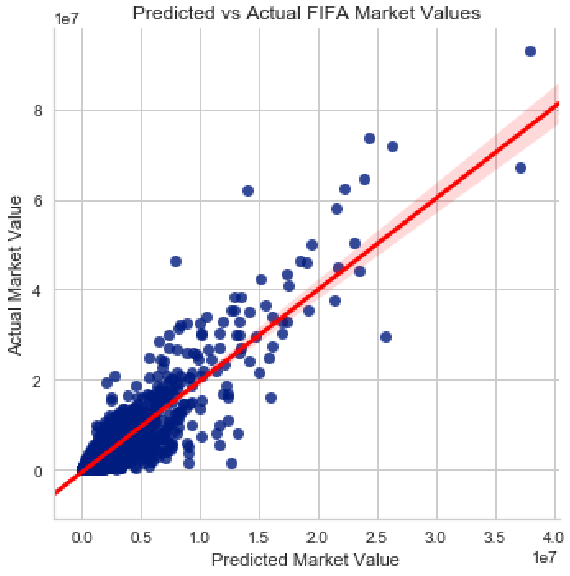
```
Linear Regression R squared": 0.7702
```

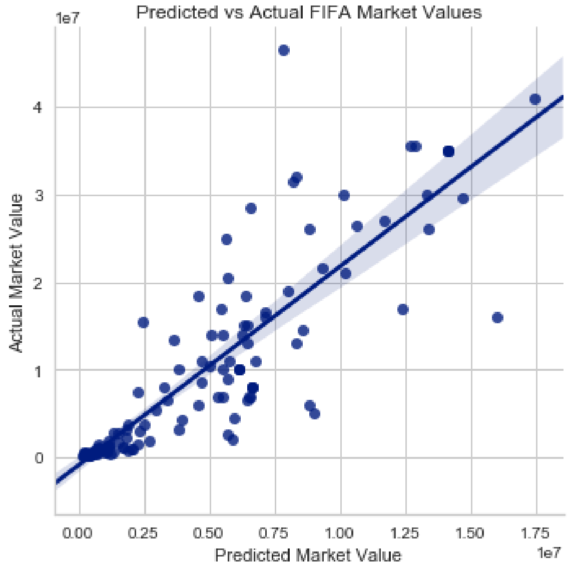
```
from sklearn.metrics import max_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import mean_absolute_error

print('Explained Variance Score: %.4f' % explained_variance_score(y_test, pred))
print('Max Error: %.4f' % max_error(y_test, pred))
print('Mean Absolute Error: %.4f' % mean_absolute_error(y_test, pred))
print('Mean Squared Error: %.4f' % metrics.mean_squared_error(y_test, pred))
print('Root Mean Squared Error: %.4f' %  np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
Explained Variance Score: 0.7702
Max Error: 3.7325
Mean Absolute Error: 0.4973
Mean Squared Error: 0.4280
Root Mean Squared Error: 0.6542
```

Once the model was split, the data was fitted to a linear regression model and predictions were executed. We can see from the resulting Ordinary Least Squares Regression Results that the R-squared value was 0.771, which can be classified as a loose regression fit.

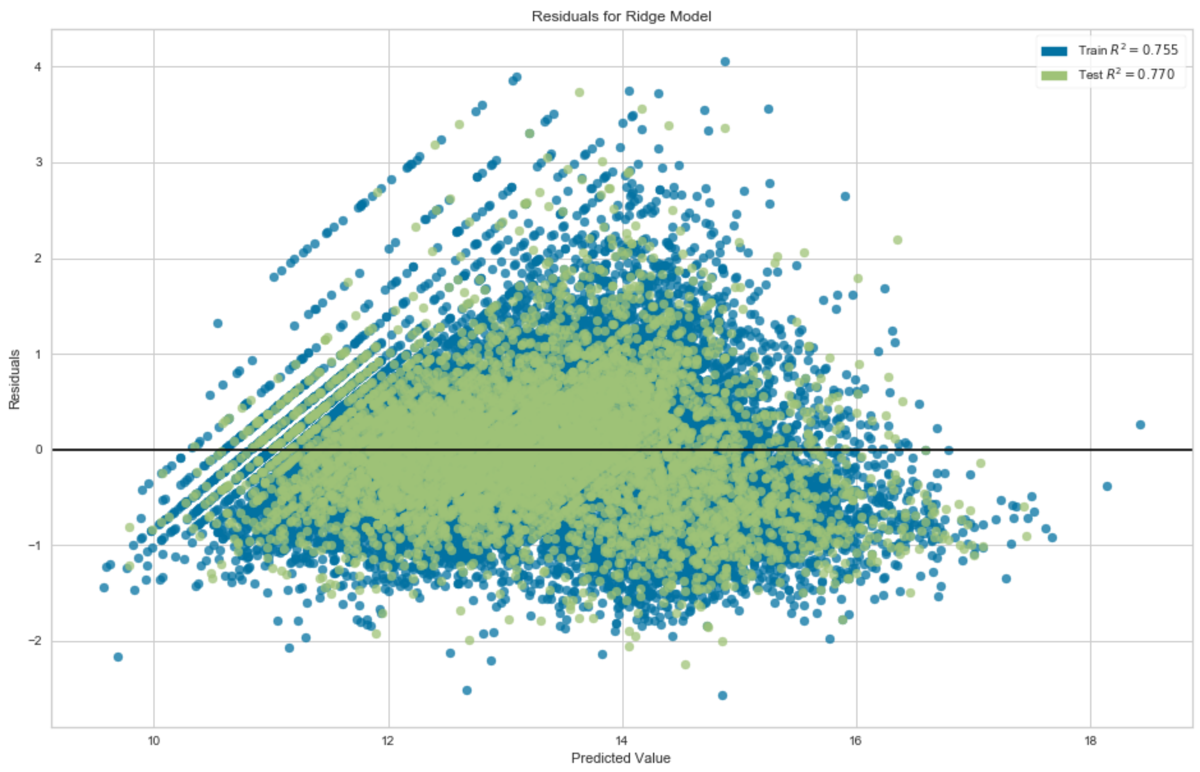
```
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_value   R-squared:                       0.771
Model:                            OLS   Adj. R-squared:                  0.770
Method:                 Least Squares   F-statistic:                     3523.
Date:                Fri, 15 Nov 2019   Prob (F-statistic):               0.00
Time:                        16:23:18   Log-Likelihood:                -6263.9
No. Observations:                6302   AIC:                         1.254e+04
Df Residuals:                    6295   BIC:                         1.259e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.4756      0.009   1504.356      0.000      13.458      13.493
pc1           -0.1896      0.002    -87.258      0.000      -0.194      -0.185
pc2            0.0565      0.004     14.598      0.000       0.049       0.064
pc3            0.4217      0.006     74.585      0.000       0.411       0.433
pc4           -0.1353      0.007    -18.930      0.000      -0.149      -0.121
pc5            0.5879      0.008     78.044      0.000       0.573       0.603
pc6            0.0158      0.012      1.277      0.202      -0.008       0.040
==============================================================================
Omnibus:                      636.284   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1332.066
Skew:                          -0.643   Prob(JB):                    5.57e-290
Kurtosis:                       4.849   Cond. No.                         6.07
==============================================================================
```



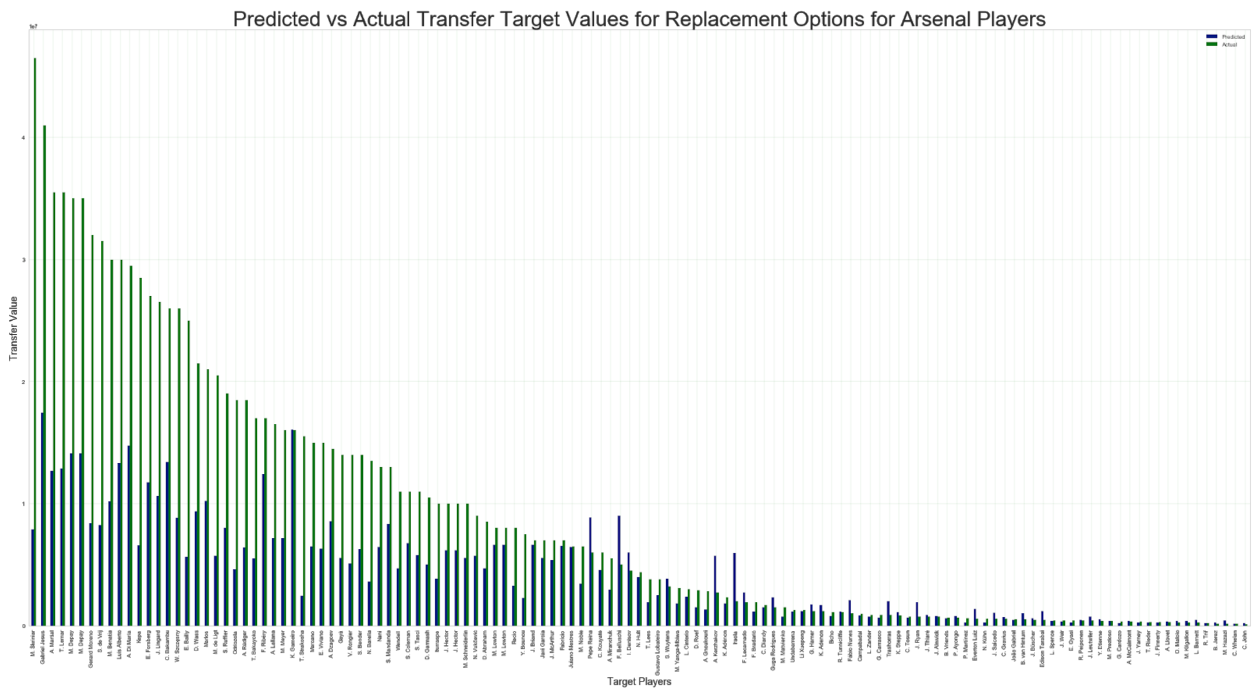


The entire test data set and the selected scouted players follow the same trend, loosely following the actual market values of the players. The consistency of the model between the subset and entire data set provides insights how overpredicions or underpredictions could have occured.

It was observed that the residuals between the predicted values and actual values roughly fit the same trends according to a Ridge model.



It was also observed that extremely higher valued players were underpredicted while lower valued player values were predicted closer to their actual valuation.



XGBoost Linear Regression

XGBoost Linear Regression methods differed from scikit-learn linear regression methods however the concept remained the same. A selection was parsed out and the training data set consisted of all PCA components plus the ‘log_value’ column, while the test data set consisted of only the PCA components.

```
import xgboost as xgb
Train_Master = df_pca.drop(['value_eur', 'club', 'short_name'], axis=1)
Test_Master = df_pca.drop(['log_value','value_eur', 'club', 'short_name'], axis=1)
```

```
Train_Master.shape, Test_Master.shape
```

```
((31023, 7), (31023, 6))
```

```
Train, Test = train_test_split(Train_Master[0:100000], test_size = 0.2)
```

```
X_train = Train.drop(['log_value'], axis=1)
Y_train = Train["log_value"]
X_test = Test.drop(['log_value'], axis=1)
Y_test = Test["log_value"]
```

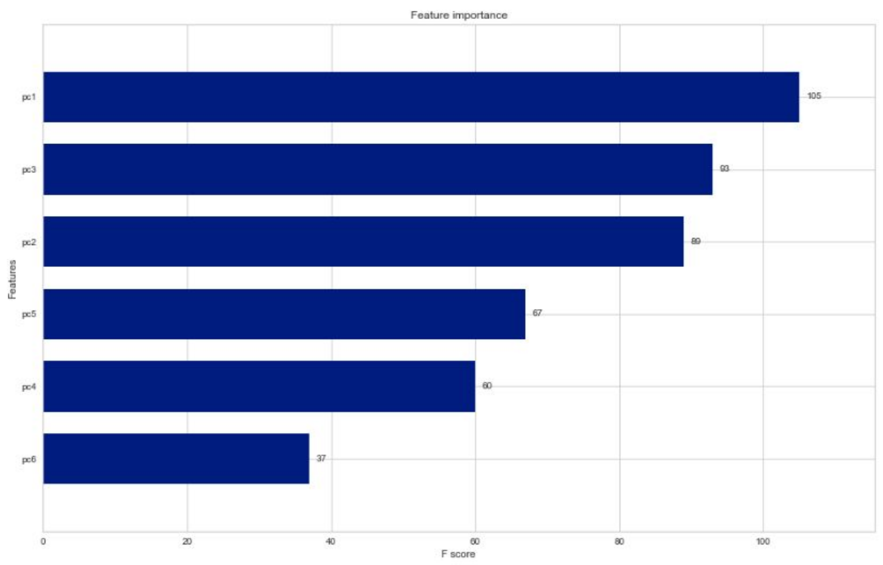
```
Test_Selection = Test_Master.loc[Test_Master.index.isin(selection)]
```

```
dtrain = xgb.DMatrix(X_train, label=Y_train)
dvalid = xgb.DMatrix(X_test, label=Y_test)
dtest = xgb.DMatrix(Test_Selection)
watchlist = [(dtrain, 'train'), (dvalid, 'valid')]
```

It can be observed that when the PCA dataset was executed using XGBoost, the modeling RMSE was equivalent to 0.59. It can also be observed that the PCA components’ F-Score values within the model decrease from the first PCA component being the most important while the sixth PCA component is the least important, as expected.

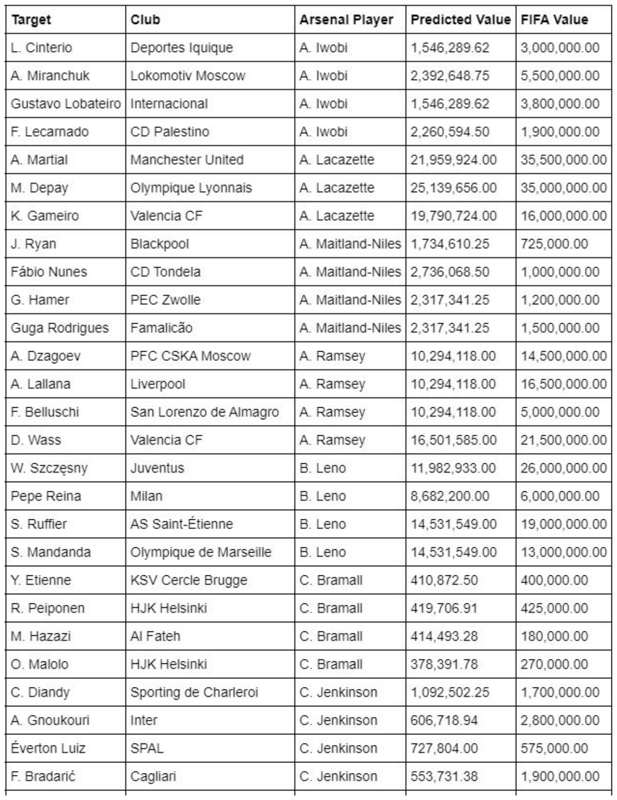
```
xgb_pars = {'min_child_weight': 1, 'eta': 0.5, 'colsample_bytree': 0.9,
            'max_depth': 6,
'subsample': 0.9, 'lambda': 1., 'nthread': -1, 'booster' : 'gbtree', 'silent': 1,
'eval_metric': 'rmse', 'objective': 'reg:linear'}
model = xgb.train(xgb_pars, dtrain, 10, watchlist, early_stopping_rounds=2,
      maximize=False, verbose_eval=1)
print('Modeling RMSLE %.5f' % model.best_score)
```

```
[0]     train-rmse:6.44853      valid-rmse:6.46899
Multiple eval metrics have been passed: 'valid-rmse' will be used for early stopping.

Will train until valid-rmse hasn't improved in 2 rounds.
[1]     train-rmse:3.27905      valid-rmse:3.29409
[2]     train-rmse:1.73508      valid-rmse:1.75306
[3]     train-rmse:1.02226      valid-rmse:1.04755
[4]     train-rmse:0.716965     valid-rmse:0.745426
[5]     train-rmse:0.610949     valid-rmse:0.64388
[6]     train-rmse:0.574743     valid-rmse:0.611303
[7]     train-rmse:0.561907     valid-rmse:0.599476
[8]     train-rmse:0.555035     valid-rmse:0.596806
[9]     train-rmse:0.550756     valid-rmse:0.593697
Modeling RMSLE 0.59370
```



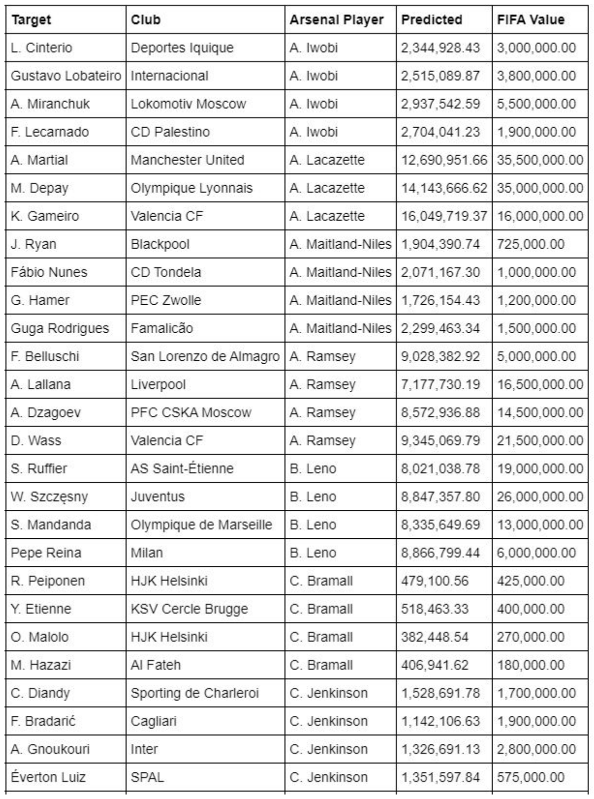
Scouting Report of Recommended Players and Predicted Values

Scikit-learn Linear Regression Results



Target	Club	Arsenal Player	Predicted Value	FIFA Value
L. Cinterio	Deportes Iquique	A. Iwobi	1,546,289.62	3,000,000.00
A. Miranchuk	Lokomotiv Moscow	A. Iwobi	2,392,648.75	5,500,000.00
Gustavo Lobateiro	Internacional	A. Iwobi	1,546,289.62	3,800,000.00
F. Lecarnado	CD Palestino	A. Iwobi	2,260,594.50	1,900,000.00
A. Martial	Manchester United	A. Lacazette	21,959,924.00	35,500,000.00
M. Depay	Olympique Lyonnais	A. Lacazette	25,139,656.00	35,000,000.00
K. Gameiro	Valencia CF	A. Lacazette	19,790,724.00	16,000,000.00
J. Ryan	Blackpool	A. Maitland-Niles	1,734,610.25	725,000.00
Fábio Nunes	CD Tondela	A. Maitland-Niles	2,736,068.50	1,000,000.00
G. Hamer	PEC Zwolle	A. Maitland-Niles	2,317,341.25	1,200,000.00
Guga Rodrigues	Famalicão	A. Maitland-Niles	2,317,341.25	1,500,000.00
A. Dzagoev	PFC CSKA Moscow	A. Ramsey	10,294,118.00	14,500,000.00
A. Lallana	Liverpool	A. Ramsey	10,294,118.00	16,500,000.00
F. Belluschi	San Lorenzo de Almagro	A. Ramsey	10,294,118.00	5,000,000.00
D. Wass	Valencia CF	A. Ramsey	16,501,585.00	21,500,000.00
W. Szczęsny	Juventus	B. Leno	11,982,933.00	26,000,000.00
Pepe Reina	Milan	B. Leno	8,682,200.00	6,000,000.00
S. Ruffier	AS Saint-Étienne	B. Leno	14,531,549.00	19,000,000.00
S. Mandanda	Olympique de Marseille	B. Leno	14,531,549.00	13,000,000.00
Y. Etienne	KSV Cercle Brugge	C. Bramall	410,872.50	400,000.00
R. Peiponen	HJK Helsinki	C. Bramall	419,706.91	425,000.00
M. Hazazi	Al Fateh	C. Bramall	414,493.28	180,000.00
O. Malolo	HJK Helsinki	C. Bramall	378,391.78	270,000.00
C. Diandy	Sporting de Charleroi	C. Jenkinson	1,092,502.25	1,700,000.00
A. Gnoukouri	Inter	C. Jenkinson	606,718.94	2,800,000.00
Éverton Luiz	SPAL	C. Jenkinson	727,804.00	575,000.00
F. Bradarić	Cagliari	C. Jenkinson	553,731.38	1,900,000.00

XGBoost Linear Regression Results



Target	Club	Arsenal Player	Predicted	FIFA Value
L. Cinterio	Deportes Iquique	A. Iwobi	2,344,928.43	3,000,000.00
Gustavo Lobateiro	Internacional	A. Iwobi	2,515,089.87	3,800,000.00
A. Miranchuk	Lokomotiv Moscow	A. Iwobi	2,937,542.59	5,500,000.00
F. Lecarnado	CD Palestino	A. Iwobi	2,704,041.23	1,900,000.00
A. Martial	Manchester United	A. Lacazette	12,690,951.66	35,500,000.00
M. Depay	Olympique Lyonnais	A. Lacazette	14,143,666.62	35,000,000.00
K. Gameiro	Valencia CF	A. Lacazette	16,049,719.37	16,000,000.00
J. Ryan	Blackpool	A. Maitland-Niles	1,904,390.74	725,000.00
Fábio Nunes	CD Tondela	A. Maitland-Niles	2,071,167.30	1,000,000.00
G. Hamer	PEC Zwolle	A. Maitland-Niles	1,726,154.43	1,200,000.00
Guga Rodrigues	Famalicão	A. Maitland-Niles	2,299,463.34	1,500,000.00
F. Belluschi	San Lorenzo de Almagro	A. Ramsey	9,028,382.92	5,000,000.00
A. Lallana	Liverpool	A. Ramsey	7,177,730.19	16,500,000.00
A. Dzagoev	PFC CSKA Moscow	A. Ramsey	8,572,936.88	14,500,000.00
D. Wass	Valencia CF	A. Ramsey	9,345,069.79	21,500,000.00
S. Ruffier	AS Saint-Étienne	B. Leno	8,021,038.78	19,000,000.00
W. Szczęsny	Juventus	B. Leno	8,847,357.80	26,000,000.00
S. Mandanda	Olympique de Marseille	B. Leno	8,335,649.69	13,000,000.00
Pepe Reina	Milan	B. Leno	8,866,799.44	6,000,000.00
R. Peiponen	HJK Helsinki	C. Bramall	479,100.56	425,000.00
Y. Etienne	KSV Cercle Brugge	C. Bramall	518,463.33	400,000.00
O. Malolo	HJK Helsinki	C. Bramall	382,448.54	270,000.00
M. Hazazi	Al Fateh	C. Bramall	406,941.62	180,000.00
C. Diandy	Sporting de Charleroi	C. Jenkinson	1,528,691.78	1,700,000.00
F. Bradarić	Cagliari	C. Jenkinson	1,142,106.63	1,900,000.00
A. Gnoukouri	Inter	C. Jenkinson	1,326,691.13	2,800,000.00
Éverton Luiz	SPAL	C. Jenkinson	1,351,597.84	575,000.00

Finally the scouting report was compiled with the target player, club, Arsenal player that the target player could replace, predicted value, and actual FIFA value. Many of the values seemed to fall within a reasonable economic range when looking at predicted values compared to actual values.

Conclusion

Based on the two linear regression models and statistical evidence provided, we can reject the null hypothesis that there is no relationship between player values and skill level. There is at least some relationship between player valuation and skill level, however, there does seem to be inflation of transfer values for specific players.

This inflation in extremely high valued players can be accounted for in modeling by categorizing quality tiers of players to limit modeling outside of the tier that a current player exists in.
K-means clustering demonstrated in this project that we could accurately predict a player’s general position, but we could also cluster high valued players, average valued players, and lower valued players together, and run regression analyses on the specific tier of player to possibly eliminate the inflation component we observed in this project.