The Effect of Finishing Efficiency on Transfer Market Valuation

1. Introduction

Football, also known as soccer, is the most widely followed sport globally, played and watched across continents. Most countries have professional league systems, often with multiple divisions, where clubs compete in seasonal competitions. At the top of these systems are elite domestic leagues such as the English Premier League, Spain's La Liga, Germany's Bundesliga, Italy's Serie A, and France's Ligue 1. These leagues attract top talent from around the globe and serve as platforms where clubs not only compete for national titles but also qualify for prestigious international tournaments like the UEFA Champions League. Players' performance in these leagues often influences their recognition, career progression, and market value.

Transfers are the heartbeat of football, allowing clubs to strengthen their squads by bringing in new talent. Held during the summer and winter transfer windows, these negotiations see teams competing to sign the next big star, with player values constantly shifting based on performance, potential, league quality, and club reputation. As highlighted by FIFA in a recent report, the global transfer market has reached unprecedented heights, with the number of international transfers exceeding 5,000 for the first time and a total spend of USD 2.35 billion in a single January window (FIFA, 2025). Among all positions, attackers, especially strikers and wingers, often have the highest market value since their goals and creativity can define a team's success. However, beyond the excitement of high-profile transfers, scouting plays a crucial role in identifying talent, helping clubs find hidden gems and avoid overpaying for players. Previous studies have highlighted how player valuations fluctuate according to factors like goals per 90, key passes per 90 minutes played average, etc, and have argued that scouting is shifting toward data-driven models (Ian et al., 2023; Raffaele et al., 2024).

Clubs like Liverpool and Brighton have mastered data-driven scouting, using advanced metrics to uncover players whose true value goes beyond reputation and media hype. In a transfer market often swayed by instincts and eye-catching performances, analytics holds the key to discovering talent that often goes unnoticed. To better understand the relationship between attackers' performance and market value, this study focuses on a key metric: expected goals (xG), specifically examining how attackers' overperformance or underperformance relative to xG influences their market valuation.

2. Research Question

Does overperforming expected goals (xG) correlate with higher market valuation among attackers in professional football?

3. Background

In football, goals are the most decisive factor in determining the outcome of matches. Naturally, players who consistently score goals tend to gain more recognition, demand higher wages, and attract larger transfer fees. However, scoring a goal is not purely about the number of shots taken, it's closely tied to a player's finishing ability, which refers to how effectively a player can convert goal-scoring opportunities into actual goals.

Finishing ability has traditionally been evaluated through raw statistics such as goals scored and shot accuracy. However, these metrics often lack context; for instance, a tap-in from two yards and a shot from 30 yards count the same in goal statistics, though they differ significantly in difficulty. To address this limitation, football analytics has evolved to develop more sophisticated metrics, most notably, the concept of Expected Goals (xG), to quantify the quality of chances and more accurately assess a player's finishing performance.

3.1 Expected Goals (xG)

Uncertainty is a defining feature of sports and one of the reasons fans are so emotionally invested in them. The element of chance, knowing that luck, alongside performance, can shape the outcome, adds suspense and drama. This is especially true in football, a low-scoring game where a single moment can decide the result. Because of this, measuring performance accurately becomes challenging, prompting the development of advanced metrics to capture what traditional statistics often miss.

One such metric is expected goals, commonly abbreviated as xG. Introduced to account for the unpredictable nature of scoring in football, xG provides a probabilistic estimate of how likely a given shot is to result in a goal. Each shot is assigned a value between 0 and 1, where 0 means no chance of scoring and 1 indicates a certain goal. (Mead et al., 2023). It estimates the probability of a given shot resulting in a goal based on several features about the shot, such as distance from the shooter to the goal or the body part used by the shooter (Scholtes & Karakuş, 2024). It is a statistical metric that estimates the probability of a shot resulting in a goal, based on the characteristics of the shot and the context in which it was taken. The xG value for each shot is calculated using historical data from thousands of past shots, analyzing how often shots from similar situations resulted in goals. These situations are assessed based on several factors, such as:

- ❖ Distance from goal
- ❖ Angle of the shot
- ❖ Type of assist (through ball, cross, rebound, etc.)
- ❖ Body parts used (head, foot, etc.)
- ❖ Pressure from defenders
- ❖ Type of play (open play, set piece, counterattack, etc.)

For example, a close-range shot with no defenders around and a clear view of the goal might have an xG of 0.8, meaning, based on historical data, it has an 80% chance of being converted into a goal. A long-range shot from outside the box under defensive pressure might have an xG of 0.05 or lower.

3.2 Overperformance and Underperformance

A player is said to overperform xG when they score more goals than expected, suggesting they are converting difficult chances or finishing with a high degree of skill. Conversely, underperformance implies that the player is scoring fewer goals than expected, possibly due to poor finishing or lack of composure.

Overperformance may indicate clinical finishing, positioning intelligence, or even an element of luck. However, extreme overperformance is often difficult to sustain over long periods. Studies have shown that finishing efficiency tends to regress toward the mean over time, especially if the overperformance is due to random variance rather than repeatable skill.

When a player consistently scores more goals than their cumulative xG suggests, they are said to be overperforming their xG. This overperformance is often interpreted as a sign of high finishing quality; players who can consistently convert difficult chances into goals may possess exceptional composure, shot placement, or decision-making skills inside the box. For example, elite finishers like Lionel Messi or Harry Kane have shown tendencies to outperform xG in multiple seasons, indicating a repeatable skill rather than random variation (Anderson & Sally, 2013).

However, xG overperformance can also be influenced by factors unrelated to finishing ability, such as deflections, goalkeeper errors, or simply good fortune. A few lucky bounces or one-off long-range goals can inflate a player’s goal tally in a small sample size, leading to an overestimation of their actual skill. Conversely, players underperforming xG, scoring fewer goals than expected, might not necessarily lack quality; they could be experiencing a period of poor luck, facing top goalkeepers, or dealing with confidence issues. Over time, these fluctuations tend to balance out.

This phenomenon is supported by the statistical principle of regression toward the mean, which suggests that extremely high or low performances often move closer to average levels in subsequent periods. In football analytics, multiple studies have confirmed that finishing efficiency tends to regress, meaning that xG overperformance is often unsustainable across full seasons unless a player possesses rare and consistent shooting talent (Lucey et al., 2014; Scholtes & Karakuş, 2024).

4. Data

The dataset was compiled from two major sources: FBref and Transfermarkt. Data scraping was conducted using Python libraries, including pandas, BeautifulSoup, and selenium.

Player performance statistics were scraped from FBref, specifically from the scouting report pages of individual players. These statistics include variables such as 'Goals - xG', 'Goal-Creating Actions', 'Age', 'Minutes Played', 'League', 'Progressive Passes', 'Touches in Attacking Penalty Area', among others. To ensure the analysis focused exclusively on attackers, data was scraped from FBref by filtering for players whose listed positions were either "**FW**" or "**FW/MF**".

The scraping methodology involved the following steps:

- Extracting URLs of all clubs from six top European leagues: the Premier League, Bundesliga, La Liga, Serie A, Ligue 1, and Eredivisie.
- From each club page, retrieving the URLs of players identified as attackers.
- Accessing each player's scouting report page to extract the relevant statistics.
- Saving individual player data into separate .csv files and later merging them into a single dataset for analysis.

Market value, age, and minutes played were scraped from Transfermarkt using a similar approach: starting from club pages, navigating to player profiles, and retrieving the required information using BeautifulSoup and selenium.

After scraping the data from Transfermarkt, it was merged with the FBref dataset using the player name column as the common key. Care was taken to ensure that naming inconsistencies and duplicates were handled appropriately to align the two sources accurately. The final dataset is cross-sectional in nature, capturing player statistics and market values at a single point in time, rather than across multiple seasons.

Name	League	Age	Market Value (€M)
Vinicius Junior	La Liga	24	200
Erling Haaland	EPL	24	200
Lamine Yamal	La Liga	17	180
Kylian Mbappe	La Liga	26	170
Bukayo Saka	EPL	23	150
Florian Wirtz	Bundesliga	21	140
Phil Foden	EPL	24	130
Rodrygo	La Liga	24	100
Alexander Isak	EPL	25	100
Lautaro Martinez	Serie A	27	95

Table 1: Top 10 attackers with the highest market value

Name	League	Market Value (€M)	xG: Expected Goals
Goncalo Ramos	Ligue 1	45	1.04
Erling Haaland	EPL	200	0.88
Harry Kane	Bundesliga	90	0.83
Serhou Guirassy	Bundesliga	40	0.81
Marco Asensio	EPL	20	0.77
Ousmane Dembele	Ligue 1	75	0.75
Victor Boniface	Bundesliga	45	0.75
Marko Arnautovic	Serie A	3.5	0.72
Kylian Mbappe	La Liga	170	0.71
Mohamed Salah	EPL	55	0.69

Table 2: Top 10 attackers with the highest xG per 90

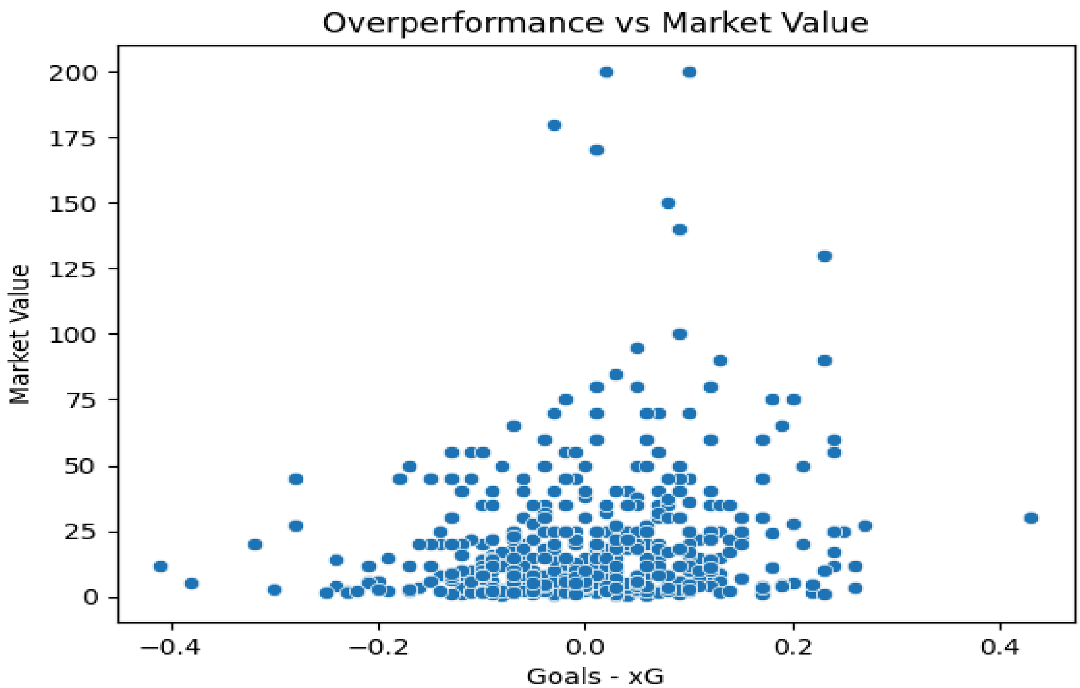
5. Methodology

To examine the impact of finishing efficiency on player market valuation, a derived metric titled **“Goals-xG”** was constructed. This metric captures the difference between a player's actual goals scored and their expected goals (xG), serving as an indicator of overperformance (positive values) or underperformance (negative values) relative to expected goal output.

An additional variable, **"League_Strength,"** was introduced to account for variations in league competitiveness. This variable was based on UEFA league coefficient values, which were normalized using a min-max scaling approach to yield comparable strength scores across leagues.

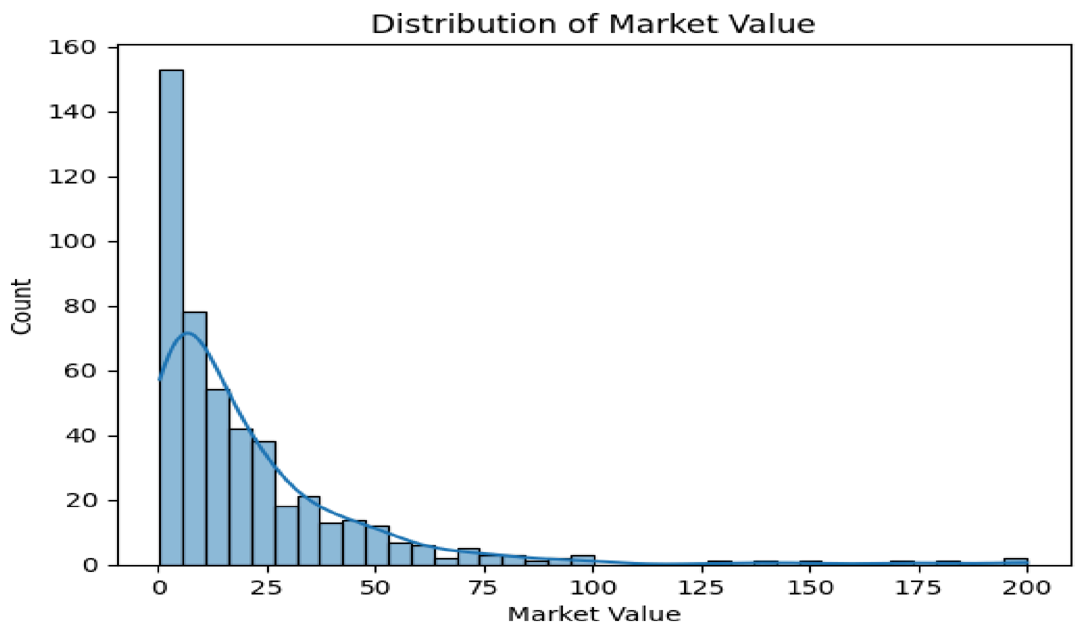
To maintain reliability, the dataset was filtered to include only players who had participated in at least 30% of the total available playing time in their respective leagues during the season. This threshold, applied using data sourced from Transfermarkt, ensured the exclusion of fringe players whose limited minutes would not support meaningful statistical analysis.

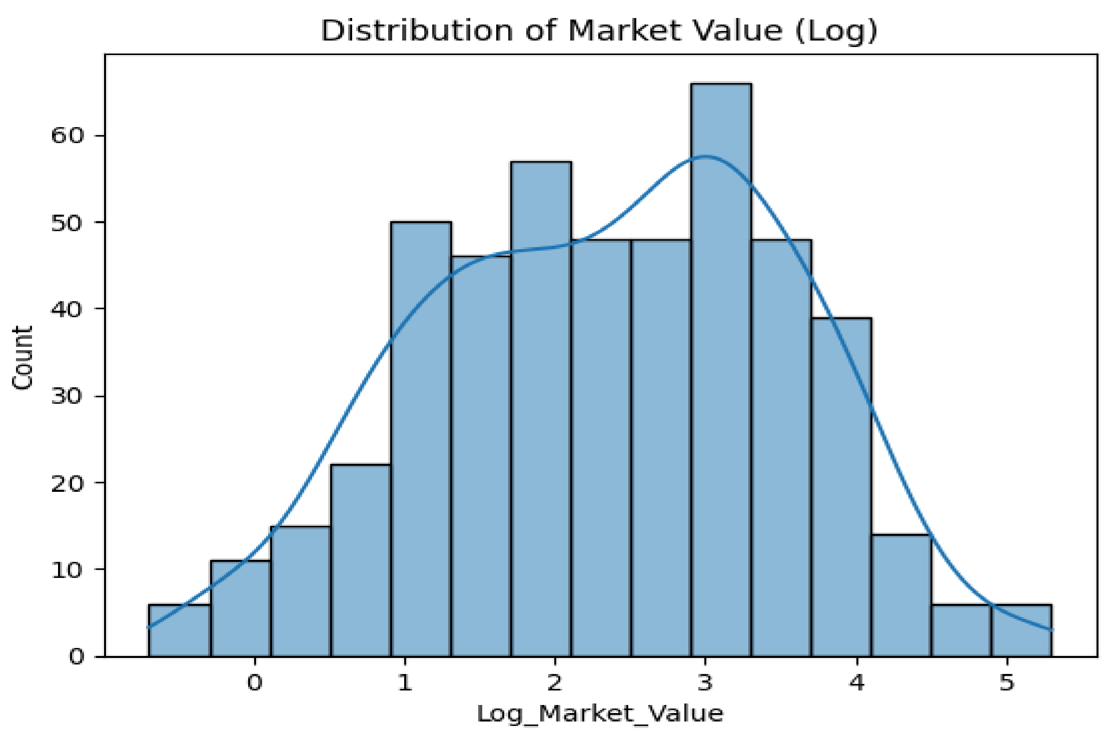
The primary objective of this analysis is to determine whether a relationship exists between finishing efficiency, captured by the Goals-xG metric, and a player's market value. To explore this, a scatter plot was generated with Goals minus xG on the x-axis and market value on the y-axis. This visualization was used to observe potential patterns and assess the correlation between the two variables.



The graph does not clearly show a relationship between xG overperformance and market value. To explore this further, a regression analysis was run using the Goals-xG variable and market value while also controlling for other relevant factors.

Since the market value distribution was uneven and showed wide variation, a log transformation was applied to bring the values into a more consistent range.





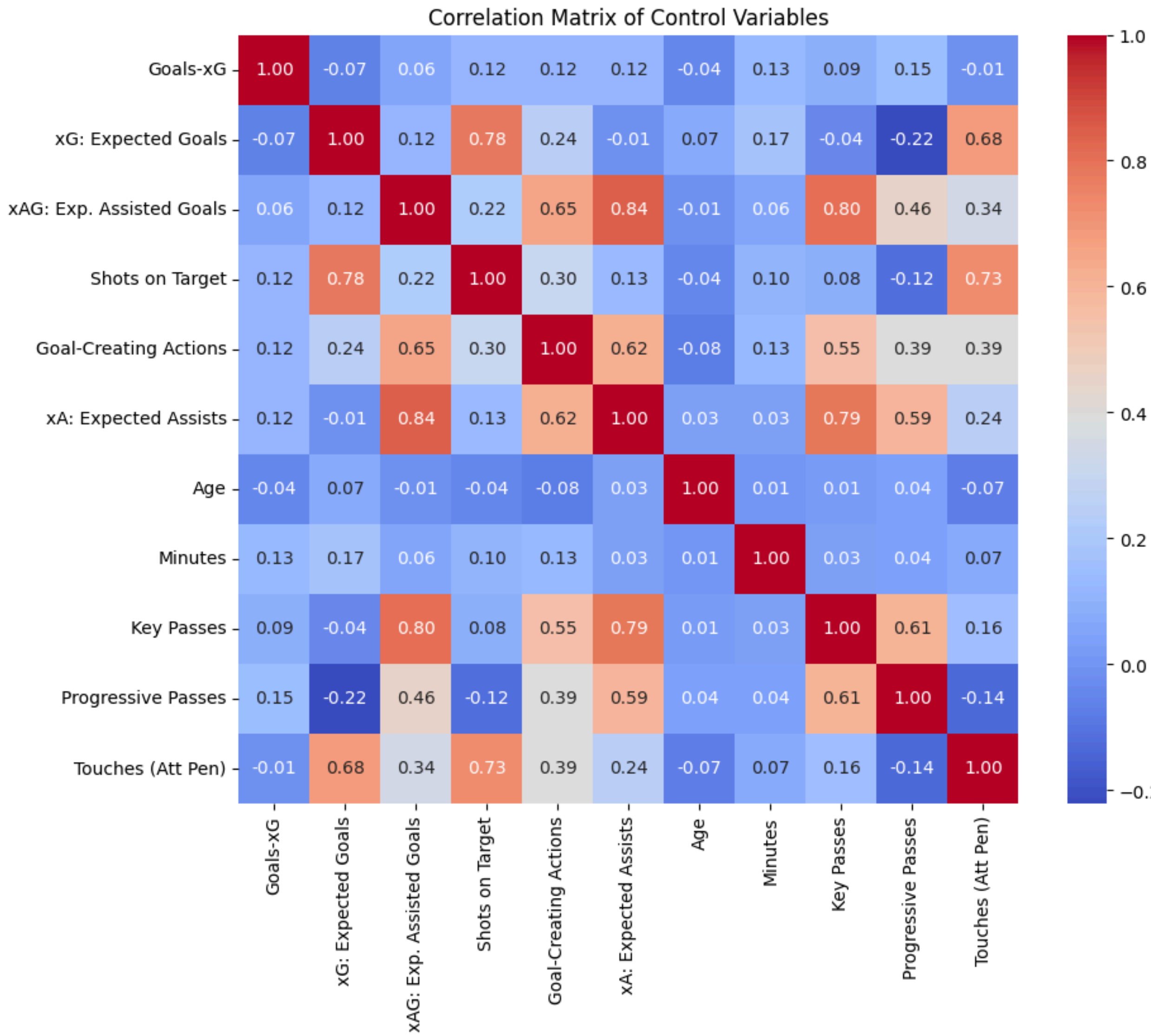
As the figure above shows, the log transformation helped make the distribution of transfer market values more even and less skewed.

5.1 Selection of Key Variables

The data was scraped from FBref and Transfermarkt, including variables such as Name, League, Goals, Assists, Non-Penalty Goals, xG (Expected Goals), npxG (Non-Penalty xG), xAG (Expected Assisted Goals), Progressive Carries, Progressive Passes, Progressive Passes Received, Shots Total, Shots on Target, Goals per Shot, npxG per Shot, xA (Expected Assists), Key Passes, Through Balls, Crosses, Shot-Creating Actions, Goal-Creating Actions, Touches in the Attacking Third, Touches in the Attacking Penalty Area, Age, Market Value, Starting Eleven, Minutes Played, and Goal Involvement.

However, not all of these variables were relevant for the analysis. The focus was mainly on those directly related to goals or expected goals. After selecting the relevant ones, a correlation matrix was plotted to check for strong relationships among the variables and to identify any potential multicollinearity.

Based on the correlation matrix below, some variables, such as "Shots on Target" and "Key Passes," were removed because they showed strong correlations with other independent variables.



5.2 Linear Regression Model

To analyze the relationship between xG overperformance and market value, a multivariate linear regression model was used. In this model, "Market Value" served as the dependent variable, while several independent variables, including "Age", "League Strength", "xG: Expected Goals", and others, were considered as explanatory variables.

The final regression model can be formulated as:-

Market Value = β0 + β1 · (Goals-xG) + β2 · (xG) + β3 · (GCA) + β4 · (Age) + β5 · (Minutes) + β6 · (League Strength) + β7 · (Prog Passes) + β8 · (Touches) + ϵ

Where:

- *$\beta 0$ is the intercept,*
- *$\beta 1, \beta 2, \ldots, \beta 8$ are the coefficients for each independent variable,*
- ***Goals − xG**: Difference between goals scored and expected goals (over/underperformance)*
- ***xG**: Expected goals, showing scoring chances*
- ***GCA**: Goal-creating actions, measures creativity*
- ***Age**: Player's age*
- ***Minutes**: Total minutes played*
- ***League Strength**: Competitiveness of the league*
- ***Prog Passes**: Progressive passes*
- ***Touches**: Touches in the attacking penalty area*
- ***ϵ**: Error term*

6. Result

The following section presents the statistical results and interpretation of this analysis.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log_Market_Value   R-squared:                       0.654
Model:                            OLS   Adj. R-squared:                  0.648
Method:                 Least Squares   F-statistic:                     111.3
Date:                Wed, 30 Apr 2025   Prob (F-statistic):          1.36e-103
Time:                        16:56:48   Log-Likelihood:                -518.72
No. Observations:                 481   AIC:                             1055.
Df Residuals:                     472   BIC:                             1093.
Df Model:                           8
Covariance Type:            nonrobust
===========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                     2.6070      0.283      9.200      0.000       2.050       3.164
Goals-xG                  1.3379      0.311      4.308      0.000       0.728       1.948
xG: Expected Goals        1.9690      0.285      6.913      0.000       1.409       2.529
Goal-Creating Actions     1.1433      0.214      5.338      0.000       0.722       1.564
Age                      -0.1394      0.009    -15.485      0.000      -0.157      -0.122
Minutes                   0.0121      0.002      6.277      0.000       0.008       0.016
League_Strength           1.1299      0.106     10.650      0.000       0.921       1.338
Progressive Passes        0.1819      0.026      7.037      0.000       0.131       0.233
Touches (Att Pen)         0.1457      0.030      4.820      0.000       0.086       0.205
==============================================================================
Omnibus:                        0.479   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.304
Skew:                           0.012   Prob(JB):                        0.859
Kurtosis:                       3.121   Cond. No.                         643.
==============================================================================
```

The results of the multivariate linear regression model indicate that approximately 65.4% of the variance in log-transformed market values can be explained by the included predictors ($R^2 = 0.654$, $p < 0.001$), suggesting a good overall model fit.

The primary variable of interest, Goals minus xG, has a significant and positive coefficient ($\beta = 1.34$, $p < 0.001$). This suggests that players who consistently outperform their expected goals are likely to be more highly valued in the transfer market.

Notably, xG itself exhibits the strongest effect among all predictors ($\beta = 1.969$, $p < 0.001$), reinforcing that players who regularly get into high-quality scoring positions are valued more, even beyond their finishing. Goal-Creating Actions also show a significant positive relationship ($\beta = 1.14$), underscoring the value placed on playmaking ability.

Age demonstrates a negative association ($\beta = –0.139$), indicating that older players generally see diminished market value, likely due to reduced long-term potential and a shorter career horizon. Other variables, such as Minutes played, Touches in the Attacking Penalty Area, and League Strength, are also positively associated, indicating that consistent playtime, attacking presence, and participation in competitive leagues all contribute positively to a player's market worth.

6.1 VIF Test for Multicollinearity

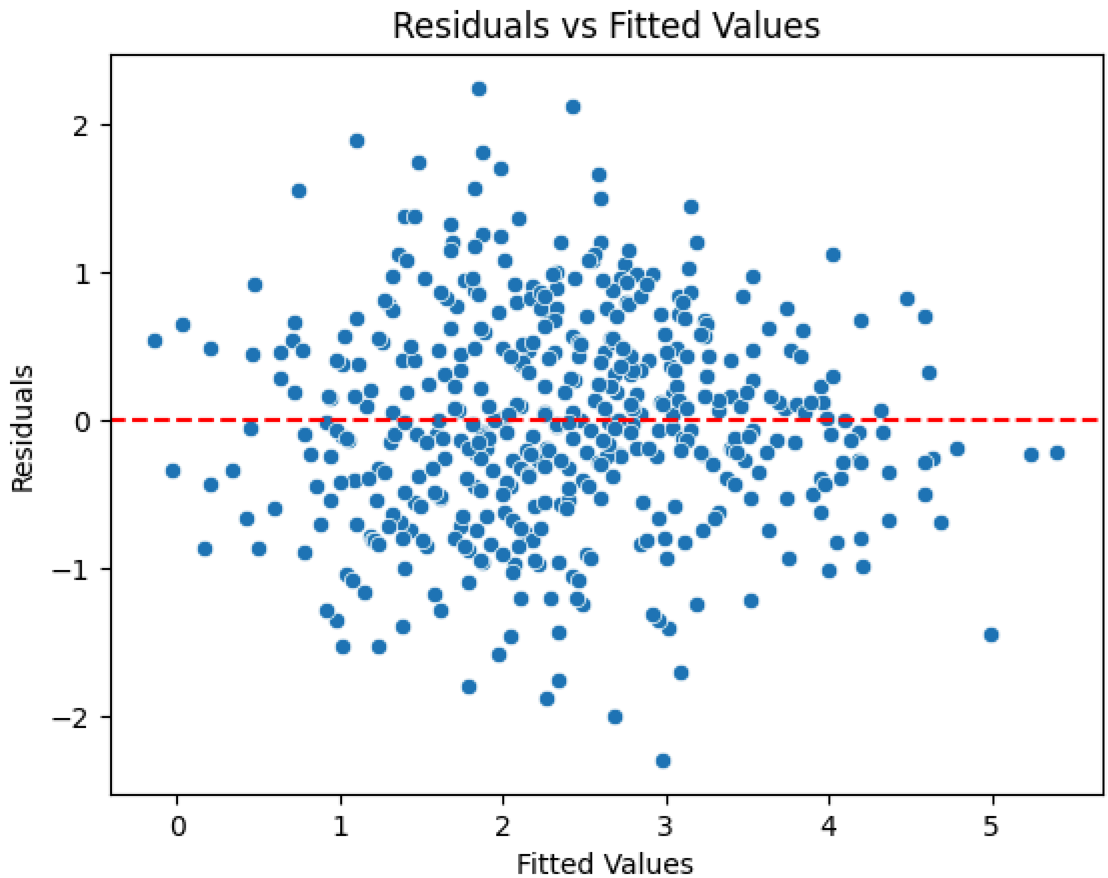
To ensure the stability and reliability of the regression estimates, multicollinearity among the independent variables was assessed using the Variance Inflation Factor (VIF). VIF values greater than 5 or 10 typically indicate problematic levels of multicollinearity.

Feature	VIF
const	74.8955681
Goals-xG	1.0501862
xG: Expected Goals	2.0943823
Goal-Creating Actions	1.5936534
Age	1.0489135
Minutes	1.0679506
League_Strength	1.0210347
Progressive Passes	1.4107938
Touches (Att Pen)	2.233812

The results show that all predictors fall well below the critical threshold, with most VIF values close to 1 and the highest being 2.23 for Touches in the Attacking Penalty Area. This suggests that multicollinearity is not a concern in the model, and the estimated coefficients are likely to be robust and interpretable.

6.2 Heteroskedasticity

The next step involves testing for heteroskedasticity, whether the variance of residuals remains constant across all levels of the independent variables. Addressing this is essential to validate the efficiency of the ordinary least squares (OLS) estimates and ensure accurate inference.



In the graph above, a residuals vs fitted values plot was generated by plotting the model's residuals on the y-axis against the fitted (predicted) values on the x-axis. The purpose of this plot is to visually assess the presence of heteroskedasticity. A horizontal red reference line at zero was added to help evaluate the dispersion of residuals around the mean.

From the plot, the residuals appear to be randomly scattered around the zero line with no clear pattern or funnel shape, suggesting that the variance of the errors remains relatively constant across different levels of predicted values. This visual evidence indicates that heteroskedasticity is unlikely to be a concern in the model.

6.3 Breusch Pagan Test

To further assess whether heteroskedasticity is present in the regression model, the Breusch-Pagan test was conducted.

This test helps identify whether the residuals from the regression model exhibit non-constant variance. Below are the results of the test:

Statistic	Value
LM Statistic	15.42
LM-Test p-value	0.0515
F-Statistic	1.95
F-Test p-value	0.0505

The results of the Breusch-Pagan test indicate that there is marginal evidence of heteroskedasticity. The p-value for both the LM test (0.0515) and the F-test (0.0505) is slightly above the conventional 0.05 significance level. This suggests that the null hypothesis of homoskedasticity cannot be rejected at the 5% significance level, but it is very close.

6.4 White test

To further investigate the presence of heteroskedasticity, the White test was performed. The White test is a more general test for heteroskedasticity and can detect both linear and non-linear forms of variance inconsistency in the residuals. Given the potential evidence of heteroskedasticity from the Breusch-Pagan test, the White test provides an additional layer of validation.

Below are the results of the White test:

Statistic	**Value**
LM Statistic	61.24
LM-Test p-value	0.0436
F-Statistic	1.45
F-Test p-value	0.0367

The results from the White test provide stronger evidence of heteroskedasticity. With both the LM-Test p-value (0.0436) and the F-Test p-value (0.0367) being below the 0.05

significance level, we can reject the null hypothesis of homoskedasticity at the 5% significance level. This suggests that the model likely suffers from heteroskedasticity

6.5 Using a Robust Model

Since both the Breusch-Pagan and White tests indicated the presence of heteroskedasticity, a robust regression model was used to correct for this issue. Robust standard errors do not change the estimated coefficients of the model but adjust the standard errors to account for non-constant variance in the residuals. This helps ensure that the significance tests for each predictor remain valid, even when heteroskedasticity is present. The summary of the robust model is given below:-

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log_Market_Value   R-squared:                       0.654
Model:                            OLS   Adj. R-squared:                  0.648
Method:                 Least Squares   F-statistic:                     141.3
Date:                Wed, 30 Apr 2025   Prob (F-statistic):         4.14e-120
Time:                        14:19:00   Log-Likelihood:                -518.72
No. Observations:                 481   AIC:                             1055.
Df Residuals:                     472   BIC:                             1093.
Df Model:                           8
Covariance Type:                  HC3
===========================================================================================
                            coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------
const                     2.6070      0.289      9.009      0.000       2.040       3.174
Goals-xG                  1.3379      0.313      4.272      0.000       0.724       1.952
xG: Expected Goals        1.9690      0.304      6.480      0.000       1.373       2.565
Goal-Creating Actions     1.1433      0.225      5.083      0.000       0.703       1.584
Age                      -0.1394      0.009    -15.610      0.000      -0.157      -0.122
Minutes                   0.0121      0.002      6.043      0.000       0.008       0.016
League_Strength           1.1299      0.098     11.476      0.000       0.937       1.323
Progressive Passes        0.1819      0.028      6.587      0.000       0.128       0.236
Touches (Att Pen)         0.1457      0.034      4.338      0.000       0.080       0.211
==============================================================================
Omnibus:                        0.479   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.304
Skew:                           0.012   Prob(JB):                        0.859
Kurtosis:                       3.121   Cond. No.                         643.
==============================================================================
```

7. Limitations of the Research

This study faced several limitations that impacted the depth of its findings. First, the final dataset was relatively small. Although the initial scrape from FBref yielded data on 701 players, applying a minimum minutes-played threshold (30 percent of total minutes) reduced the sample size to 481. This limits the statistical ability of the model.

Second, FBref only provides a snapshot of each player's scouting report over the past 365 days. As a result, metrics such as xG reflect a one-year period rather than a broader historical performance trend, constraining the ability to observe long-term patterns.

Furthermore, several potentially influential variables, such as player height, international caps (particularly relevant for younger players), and remaining contract length, were not included. These omissions likely restricted the model's ability to capture the full range of factors that contribute to market valuation.

Finally, certain qualitative factors that can influence market value, such as a player's style of play, charisma, or social media presence, are difficult to quantify and were not included. These factors can significantly affect commercial value, especially for high-profile attackers.

Expanding the dataset to include historical trends and more comprehensive variables would enhance the robustness of future analyses and provide a deeper understanding of the relationship between performance metrics and market valuation in professional football.

8. Source Code and Dataset

All code scripts and datasets used in this study are publicly available on GitHub. Please refer to the following repository: https://github.com/Kaushal-Dhungel/players-analysis

9. Acknowledgment of AI Assistance

ChatGPT (OpenAI, 2025) was used for grammatical corrections, language refinement, assistance with regression analysis, and summarizing findings in this document.

References

1. FIFA. (2025). *International transfer snapshot (January 2025)*. Retrieved from https://inside.fifa.com/transfer-system/international-transfer-snapshot

2. McHale, I. G., & Holmes, B. (2023). Estimating transfer fees of professional footballers using advanced performance metrics and machine learning. *European Journal of Operational Research, 306*(1), 389–399. https://doi.org/10.1016/j.ejor.2022.06.033

3. Poli, R., Besson, R., & Ravenel, L. (2024). Statistical modeling of football players' transfer fees worldwide. *International Journal of Financial Studies, 12*(3), 93. https://doi.org/10.3390/ijfs12030093

4. Mead, J., O'Hare, A., & McMenemy, P. (2023). Expected goals in football: Improving model performance and demonstrating value. *PLOS ONE, 18*(4), e0282295. https://doi.org/10.1371/journal.pone.0282295

5. Scholtes, A., & Karakuş, O. (2024). Bayes-xG: Player and position correction on expected goals (xG) using a Bayesian hierarchical approach. *Frontiers in Sports and Active Living, 6*, 1348983. https://doi.org/10.3389/fspor.2024.1348983

6. Anderson, C., & Sally, D. (2013). *The numbers game: Why everything you know about football is wrong*. Penguin Books.

7. Lucey, P., Bialkowski, A., Monfort, M., Carr, P., & Matthews, I. (2014). Quality vs. quantity: Improved shot prediction in soccer using strategic features from spatiotemporal data. *MIT Sloan Sports Analytics Conference*.

8. OpenAI. (2025). *ChatGPT (GPT-4)* [Large language model]. OpenAI. https://chat.openai.com

i7pabk53v

May 1, 2025

0.0.1 Code used to scrape FBREF

0.0.2 1. SCRAPE ALL LEAGUE TEAMS

[ ]:
```python
import requests
import pandas as pd
from bs4 import BeautifulSoup
import http.client
import re
import time
import json
import html5lib
import os
```

[ ]:
```python
# Scrape all league teams
fbref_league_names = {
    "pl":[9, "Premier-League", 20],
    "seriea":[11, "Serie-A", 20],
    "laliga":[12, "La-Liga", 20],
    "bundesliga":[20, "Bundesliga", 18],
    "league1":[13, "Ligue-1", 18],
    "eredivisie": [23, "Eredivisie", 19]
}

fbref_scrapped = {}
def scrape_league_teams(code, name, number):

    url = f'https://fbref.com/en/comps/{code}/{name}-Stats'

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                      "AppleWebKit/537.36 (KHTML, like Gecko) "
                      "Chrome/122.0.0.0 Safari/537.36",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,"
                  "image/avif,image/webp,image/apng,*/*;q=0.8,"
                  "application/signed-exchange;v=b3;q=0.7",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "en-US,en;q=0.9",
```

```
        "Referer": "https://www.google.com/",
        "DNT": "1",
        "Connection": "keep-alive",
        "Upgrade-Insecure-Requests": "1"
    }

    session = requests.Session()
    response = session.get(url, headers=headers)
    soup = BeautifulSoup(response.text, 'html.parser')
    anchors = soup.find_all('a', href=re.compile(r'^/en/squads/'))
    anchors = anchors[:number+1]
    fbref_scrapped[name] = anchors
    time.sleep(10)
```

[ ]:
```
## SCRAPING ALL THE LEAGUE TEAMS
for key,val in fbref_league_names.items():
    scrape_league_teams(val[0], val[1], val[2])
```

[ ]:
```
# SAVED ALL TEAMS URL TO A FILE
def extract_hrefs(data):
    result = {}
    for league, tags in data.items():
        hrefs = []
        for tag in tags:
            if hasattr(tag, 'get'):
                href = tag.get('href')
                if href:
                    hrefs.append(href)
        result[league] = hrefs
    return result

cleaned_data = extract_hrefs(fbref_scrapped)

# Write to file
with open('fbref_team_links.json', 'w') as f:
    json.dump(cleaned_data, f, indent=2)
```

[ ]:
```
```

0.0.3 2. SCRAPE ALL PLAYERS LINK FROM EACH TEAM

[ ]:
```
league_all_attackers = {}

def scrape_team_attackers(league_name,league_urls):
    all_attackers = []
    for team_url in league_urls:
        if team_url == "/en/squads/":
```

```
            continue

        url = f"https://fbref.com/{team_url}"

        headers = {
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                        "AppleWebKit/537.36 (KHTML, like Gecko) "
                        "Chrome/122.0.0.0 Safari/537.36",
            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,"
                    "image/avif,image/webp,image/apng,*/*;q=0.8,"
                    "application/signed-exchange;v=b3;q=0.7",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en;q=0.9",
            "Referer": "https://www.google.com/",
            "DNT": "1",  # Do Not Track
            "Connection": "keep-alive",
            "Upgrade-Insecure-Requests": "1"
        }

        session = requests.Session()
        response = session.get(url, headers=headers)
        soup = BeautifulSoup(response.text, 'html.parser')
        anchors = soup.find_all('a', href=re.compile(r'^/en/players/'))
        html = pd.read_html(url)
        df = html[0]
        df.columns = df.columns.get_level_values(-1)
        df = df[df["Pos"].str.contains("FW", na=False)]
        attackers_list = df["Player"].to_list()
        filtered_tags = [
            tag for tag in anchors
            if tag.text in attackers_list and "matchlogs" not in tag['href']
        ]

        all_attackers += filtered_tags

        time.sleep(10)

    league_all_attackers[league_name] = all_attackers
```

[ ]:
```
with open('fbref_team_links.json', 'r') as f:
    clubs_link_data = json.load(f)

for key,val in clubs_link_data:
    scrape_team_attackers(key, clubs_link_data[key])
```

[ ]:
```
cleaned_data = extract_hrefs(league_all_attackers)
```

```
with open('attackers/all_attackers.json', 'w') as f:
    json.dump(cleaned_data, f, indent=2)
```

[ ]:

```

```

0.0.4 3. SCRAPE PLAYERS STATS

[ ]:

```
def convert_urls_to_scouting(urls):
    base = "https://fbref.com"
    return [
        f"{base}{match.group(1)}/scout/365_m1/{match.group(2)}-Scouting-Report"
        for url in urls
        if (match := re.match(r"(/en/players/[\w\d]+)/([\w\-]+)", url))
    ]

folders = os.listdir("players")
players_name_list = []

for f in folders:
    players_name_list += [
            filename.replace(".csv", "")
            for filename in os.listdir(f"players/{f}")
            if filename.endswith(".csv")
        ]
```

[ ]:

```
def scrape_players_data(league,urls):
    urls = list(set(urls))  # remove duplicates
    urls_formatted = convert_urls_to_scouting(urls)

    for url in urls_formatted:
        last_part = url.rstrip('/').split("/")[-1]
        player_name = last_part.replace("-Scouting-Report", "").replace("-", "␣
 ↪")

        # if player_name not in rescrape:
        #     continue
        # if player_name in players_name_list:
        #     continue

        try:
            tables = pd.read_html(url)
            if len(tables) > 3:
                table = tables[-2]

            else:
                table = tables[2]
            table.to_csv(f"players/{league}/{player_name}.csv")
```

```
        except Exception as e:
            print(F"ERROR FETCHING {player_name} data", e)

        time.sleep(10)
```

[ ]:
```
with open('attackers/all_attackers.json', 'r') as f:
    players_link_data = json.load(f)

for key,val in players_link_data.items():
    scrape_players_data(key, players_link_data[key])
```

[ ]:
```

```

0.0.5 4. CLEAN AND MERGE ALL PLAYERS DATA

[ ]:
```
## THESE ARE THE METRICS I AM MOST INTERESTED IN
search_terms = [
    "Goals",
    "Assists",
    "Non-Penalty Goals",
    "xG: Expected Goals",
    "npxG: Non-Penalty xG",
    "xAG: Exp. Assisted Goals",
    "Progressive Carries",
    "Progressive Passes",
    "Progressive Passes Rec",
    "Shots Total",
    "Shots on Target",
    "Goals/Shot",
    "npxG/Shot",
    "xA: Expected Assists",
    "Key Passes",
    "Through Balls",
    "Crosses",
    "Shot-Creating Actions",
    "Goal-Creating Actions",
    "Shots on Target",
    "Touches (Att 3rd)",
    "Touches (Att Pen)",
]

whole_df = pd.DataFrame(columns=["Name", "League"] + search_terms)
whole_df.head()
```

[ ]:
```
folders = os.listdir("players")
for folder in folders:
    files = os.listdir(f"players/{folder}")
```

```python
    for file in files:
        df = pd.read_csv(f"players/{folder}/{file}")
        player_name = file.replace(".csv", "")

        player_data = {key: None for key in ["Name"] + search_terms}
        player_data["Name"] = player_name
        player_data["League"] = folder

        for term in search_terms:
            # mask = df.apply(lambda row: row.astype(str).str.contains(term,␣
↪na=False)).any(axis=1)
            mask = df.apply(lambda row: row.astype(str).str.contains(term,␣
↪case=False, na=False, regex=False)).any(axis=1)
            if any(mask):
                try:
                    first_row = df[mask].iloc[0]
                    row_val = first_row["Standard Stats.1"]
                    player_data[term] = row_val
                except Exception as e:
                    print(f"Error extracting '{term}' for {player_name}: {e}")
            else:
                print(f"{term} not found for {player_name}")

        new_row = pd.DataFrame([player_data], columns=whole_df.columns)
        whole_df = pd.concat([whole_df, new_row], ignore_index=True)
```

[ ]:
```python
whole_df.to_csv("fbref_final.csv")
```

w4zcyxtrm

May 1, 2025

[1]:
```python
import pandas as pd
import numpy as np
import json
import os
```

[15]:
```python
all_fbref_df = pd.read_csv("fbref_final.csv")
all_transfermarkt_df = pd.read_csv("transfermarkt_final.csv")
```

[16]:
```python
all_fbref_df = all_fbref_df.drop(columns=["Unnamed: 0"], axis=1)
all_transfermarkt_df = all_transfermarkt_df.drop(columns=["Unnamed: 0"], axis=1)
```

[17]:
```python
all_fbref_df = all_fbref_df.drop_duplicates()
all_transfermarkt_df = all_transfermarkt_df.drop_duplicates()
final_data = pd.merge(all_fbref_df,all_transfermarkt_df, on="Name", how="outer")
```

[18]:
```python
final_data = final_data.drop_duplicates(subset=['Name'], keep='last')
final_data.to_csv("final_data.csv")
```

[19]:
```python
len(final_data)
```

[19]:
```
701
```

89a5xaouh

May 1, 2025

[ ]:
```
import pandas as pd
import requests
import json
import re
from bs4 import BeautifulSoup
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import os
```

0.0.1 1. SCRAPE ALL CLUB LINKS

[ ]:
```
all_club_urls = {}

# this link is for PL, I updated it manually for five different leagues
url = "https://www.transfermarkt.us/premier-league/startseite/wettbewerb/GB1"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                  "AppleWebKit/537.36 (KHTML, like Gecko) "
                  "Chrome/122.0.0.0 Safari/537.36",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,"
              "image/avif,image/webp,image/apng,*/*;q=0.8,"
              "application/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "en-US,en;q=0.9",
    "Referer": "https://www.google.com/",
    "DNT": "1",
    "Connection": "keep-alive",
    "Upgrade-Insecure-Requests": "1"
}

session = requests.Session()
response = session.get(url, headers=headers)
```

[ ]:
```
soup = BeautifulSoup(response.text, 'html.parser')
anchors = soup.find_all('a', href=re.compile(r'/.*?/startseite/verein/\d+/
↪saison_id/\d+'))
```

```
hrefs = list({a['href'] for a in anchors if a.has_attr('href')})

# Again, I updated the league name manually, example :- pl, league1, bundesliga␣
↪etc
all_club_urls["pl"] = hrefs
```

[ ]:
```
with open ("transfermarkt/all_clubs_tfmkt.json", "w") as file:
    json.dump(all_club_urls, file, indent=2)
```

0.0.2 2. SCRAPE ALL PLAYERS LINK

[ ]:
```
league_players_link = {}

def scrape_players(league,hrefs):
    all_list = []
    for href in hrefs:
        url = f"https://www.transfermarkt.us{href}"
        headers = {
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                        "AppleWebKit/537.36 (KHTML, like Gecko) "
                        "Chrome/122.0.0.0 Safari/537.36",
            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,"
                    "image/avif,image/webp,image/apng,*/*;q=0.8,"
                    "application/signed-exchange;v=b3;q=0.7",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en;q=0.9",
            "Referer": "https://www.google.com/",
            "DNT": "1",  # Do Not Track
            "Connection": "keep-alive",
            "Upgrade-Insecure-Requests": "1"
        }

        session = requests.Session()
        response = session.get(url, headers=headers)

        soup = BeautifulSoup(response.text, 'html.parser')
        anchors = soup.find_all('a', href=re.compile(r'/profil/spieler/'))
        refs = list({a['href'] for a in anchors if a.has_attr('href')})
        all_list += refs

        time.sleep(4)
    league_players_link[league] = all_list
```

[ ]:
```
for key,val in all_club_urls.items():
    scrape_players(key, all_club_urls[key])
```

```
[ ]: with open("transfermarkt/all_players_link.json", "w") as file:
         json.dump(league_players_link, file, indent=2)
```

```
[ ]:
```

0.0.3 3. SCRAPE ALL PLAYERS DATA

```
[ ]: fbref_df = pd.read_csv("fbref_final.csv")
     all_fbref_players = list(fbref_df ["Name"])

     player_unique_list = []
     error_players_list = []
     missed_players_list = []

     new_columns = [
     "Name",
     "Age",
     "Playing Stats",
     "Market Value"
     ]
     all_transfermarkt_df = pd.DataFrame(columns=new_columns)
```

```
[ ]: def scrape_players_stats_selenium(link_set):
         global all_transfermarkt_df

         for link in link_set:
             player_name = link.split("/")[1]
             player_name = [i.capitalize() for i in player_name.split("-")]
             player_name = " ".join(player_name)

             if player_name in all_fbref_players:
                 if player_name in player_unique_list or player_name in␣
       ↪missed_players_list:
                     continue

                 try:
                     url = f"https://www.transfermarkt.us{link}"

                     options = Options()
                     options.add_argument("--headless=new")
                     options.add_argument("--disable-gpu")
                     options.add_argument("--window-size=1920,1080")
                     options.add_argument("--no-sandbox")
                     options.add_argument("--disable-dev-shm-usage")
                     options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0;␣
       ↪Win64; x64) "
                                          "AppleWebKit/537.36 (KHTML, like Gecko) "
```

```
                                    "Chrome/122.0.0.0 Safari/537.36")

                driver = webdriver.Chrome(options=options)
                driver.get(url)
                time.sleep(5)

                soup = BeautifulSoup(driver.page_source, 'html.parser')

                # Extract Playing Stats
                minutes = soup.find_all(class_="percentage-value")
                minutes = [span.text.strip() for span in minutes]
                playing_stats = ",".join(minutes)

                # Extract Age
                age_tag = soup.find("span", class_="data-header__content",␣
↪itemprop="birthDate")
                age = age_tag.text.strip() if age_tag else None

                # Extract Market Value
                market_tag = soup.find("a",␣
↪class_="data-header__market-value-wrapper")
                market_val = market_tag.get_text(strip=True) if market_tag else␣
↪None
                match = re.search(r"\d+\.\d+", market_val) if market_val else␣
↪None
                market_val = float(match.group()) if match else None

                driver.quit()

                new_row = {
                    "Name": player_name,
                    "Age": age,
                    "Playing Stats": playing_stats,
                    "Market Value": market_val
                }
                all_transfermarkt_df = pd.concat(
                    [all_transfermarkt_df, pd.DataFrame([new_row])],
                    ignore_index=True
                )

                player_unique_list.append(player_name)

            except Exception as e:
                print ("Exception occured for player", player_name, e)
                error_players_list.append(player_name)

        else:
```

```
            missed_players_list.append(player_name)
```

[ ]:
```
with open("transfermarkt/all_players_link.json", "r") as f:
    league_players_link = json.load(f)


for key,val in league_players_link.items():
    scrape_players_stats_selenium(league_players_link[key])
    all_transfermarkt_df.to_csv(f"transfermarkt/{key}_players_tfmkt.csv")
```

0.0.4 4. CLEAN AND MERGE ALL PLAYERS DATA

[ ]:
```
new_columns = [
"Name",
"Age",
"Playing Stats",
"Market Value"
]
all_transfermarkt_df = pd.DataFrame(columns=new_columns)
```

[ ]:
```
global all_transfermarkt_df
for file in os.listdir("transfermarkt"):
    if not file.endswith(".csv"):
        continue

    league_name = file.replace(".csv","").split("_")[0]
    df = pd.read_csv("transfermarkt/" + file)
    all_transfermarkt_df = pd.concat([all_transfermarkt_df,df],␣
 ↪ignore_index=True)
```

[ ]:
```
all_transfermarkt_df["Age"] = all_transfermarkt_df["Age"].apply(lambda x: x.
 ↪split("(")[1][:2])
all_transfermarkt_df=all_transfermarkt_df.drop(columns=["Unnamed: 0"], axis=1)
all_transfermarkt_df[['Starting Eleven', 'Minutes', 'Goal Involvement']] =␣
 ↪all_transfermarkt_df['Playing Stats'].str.split(',', expand=True)
all_transfermarkt_df = all_transfermarkt_df.drop(columns=["Playing Stats"],␣
 ↪axis=1)
all_transfermarkt_df.head()
```

[ ]:
```
all_transfermarkt_df.to_csv("transfermarkt_final.csv")
```

fmquinetd

May 1, 2025

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[2]:
```
UEFA_COEFFS = {
    "epl": 26.821,
    "laliga": 23.250,
    "seriea": 21.187,
    "bundesliga": 18.421,
    "ligue1": 16.857,
    "eredivisie": 15.250
}
```

0.0.1 1. Data

[3]:
```
df = pd.read_csv("final_data_new.csv")
df.drop(columns=["Unnamed: 0"], axis=1, inplace=True)
df = df.dropna(how='any')
```

[4]:
```
df.isnull().values.any()
```

[4]:
```
False
```

[5]:
```
df = df[df["Minutes"]>30]
```

[6]:
```
most_expensive_players = df.sort_values(ascending=False, by="Market Value")
print(most_expensive_players[['Name', 'League','Age','Market Value']].head(10))
```

```
                Name      League   Age  Market Value
675   Vinicius Junior     laliga  24.0         200.0
208     Erling Haaland       epl  24.0         200.0
394       Lamine Yamal    laliga  17.0         180.0
391      Kylian Mbappe    laliga  26.0         170.0
117        Bukayo Saka       epl  23.0         150.0
```

```
233      Florian Wirtz  bundesliga  21.0         140.0
556          Phil Foden         epl  24.0         130.0
31       Alexander Isak         epl  25.0         100.0
592             Rodrygo      laliga  24.0         100.0
397   Lautaro Martinez      seriea  27.0          95.0
```

[7]:
```python
top_xG = df.sort_values(ascending=False, by="xG: Expected Goals")
print(top_xG[['Name', 'League', 'Market Value', 'xG: Expected Goals']].head(10))
```

```
                  Name      League  Market Value  xG: Expected Goals
265      Goncalo Ramos      ligue1          45.0                1.04
208     Erling Haaland         epl         200.0                0.88
273         Harry Kane  bundesliga          90.0                0.83
620    Serhou Guirassy  bundesliga          40.0                0.81
434      Marco Asensio         epl          20.0                0.77
538    Ousmane Dembele      ligue1          75.0                0.75
672    Victor Boniface  bundesliga          45.0                0.75
442   Marko Arnautovic      seriea           3.5                0.72
391      Kylian Mbappe      laliga         170.0                0.71
487      Mohamed Salah         epl          55.0                0.69
```

0.0.2 2. Methodology

[8]:
```python
df["Goals-xG"] = df["Goals"] - df["xG: Expected Goals"]
```

[9]:
```python
scaler = MinMaxScaler()
coefficients = list(UEFA_COEFFS.values())
normalized_coeffs = scaler.fit_transform(np.array(coefficients).reshape(-1,1)).
 ↪flatten()
league_strength_normalized = dict(zip(UEFA_COEFFS.keys(), normalized_coeffs))
df['League_Strength'] = df['League'].map(league_strength_normalized)
```

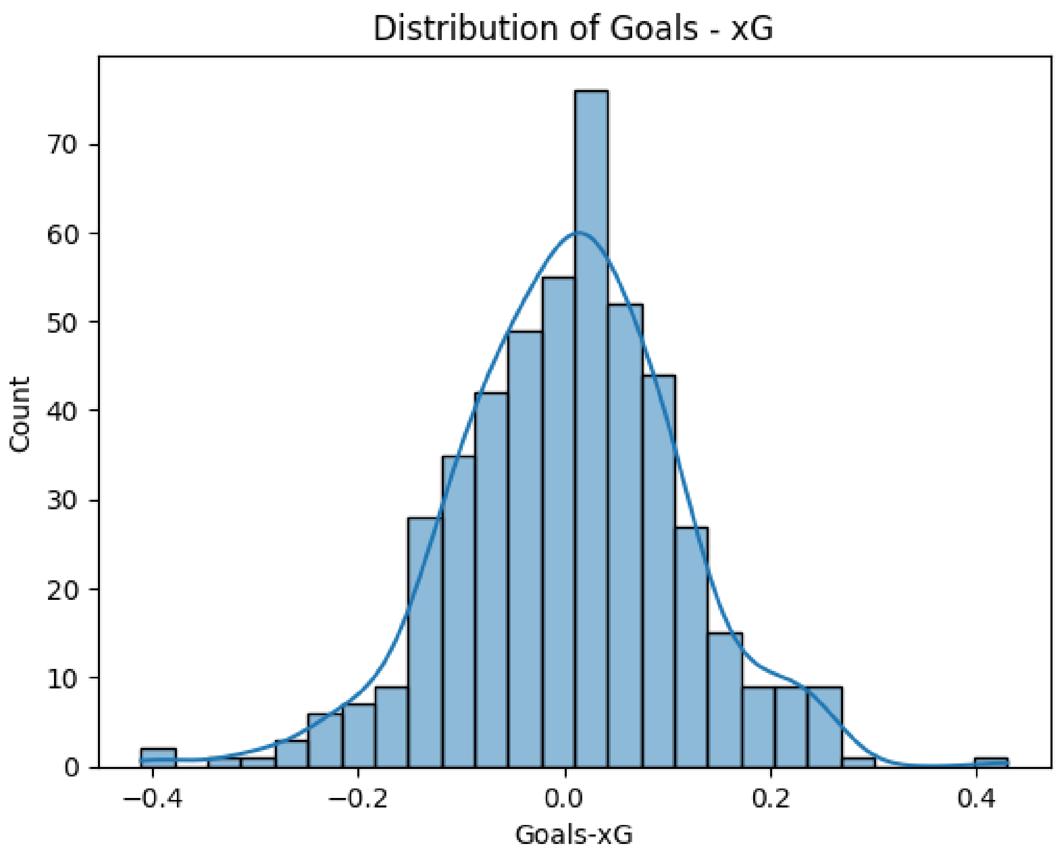
[10]:
```python
top_xg_ovp = df.sort_values(ascending=False, by="Goals-xG")
print(top_xg_ovp[['Name', 'League', 'Market Value', 'Goals-xG' ]].head(20))
```

```
                        Name      League  Market Value  Goals-xG
428            Malik Tillman  eredivisie          30.0      0.43
546            Patrik Schick  bundesliga          27.0      0.27
442         Marko Arnautovic      seriea           3.5      0.26
63          Anis Hadj Moussa  eredivisie          12.0      0.26
86               Assane Diao      laliga          25.0      0.25
452            Matheus Cunha         epl          55.0      0.24
32         Alexander Sorloth      laliga          25.0      0.24
35        Alexis Saelemaekers      seriea          17.0      0.24
9            Ademola Lookman      seriea          60.0      0.24
34     Alexis Claude Maurice  bundesliga          12.0      0.24
427            Malick Fofana      ligue1          25.0      0.24
633          Steven Skrzybski  bundesliga           1.0      0.23
```

```
359          Julian Alvarez      laliga          90.0      0.23
88              Ayoze Perez      laliga          10.0      0.23
556              Phil Foden         epl         130.0      0.23
30   Alexander Bernhardsson  bundesliga           1.5      0.22
604          Samuel Essende  bundesliga           4.5      0.22
114            Bryan Mbeumo         epl          50.0      0.21
309           Jamie Gittens  bundesliga          50.0      0.21
260        Giovani Lo Celso      laliga          20.0      0.21
```

[11]:

```python
sns.histplot(df['Goals-xG'], kde=True)
plt.title('Distribution of Goals - xG')
plt.show()
```



[12]:

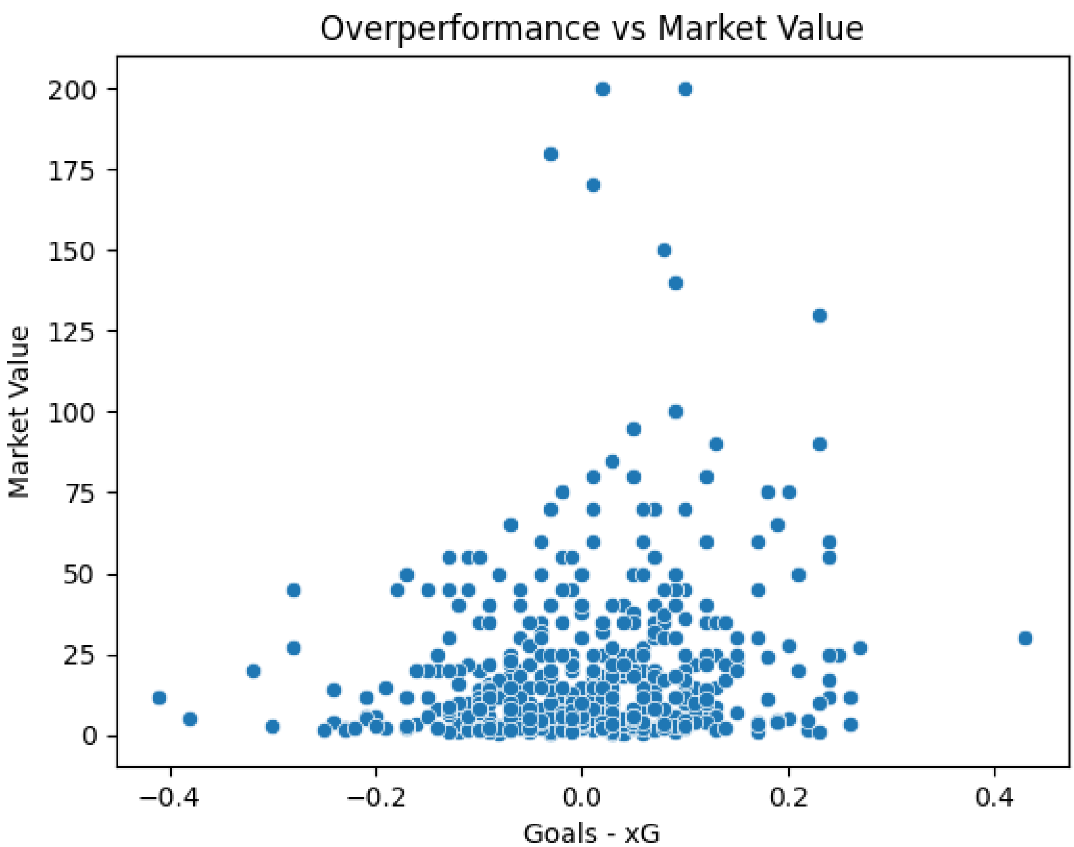
```python
print(df['Goals-xG'].describe())
```

```
count    481.000000
mean       0.004553
std        0.108174
min       -0.410000
25%       -0.060000
```

```
50%          0.010000
75%          0.070000
max          0.430000
Name: Goals-xG, dtype: float64
```

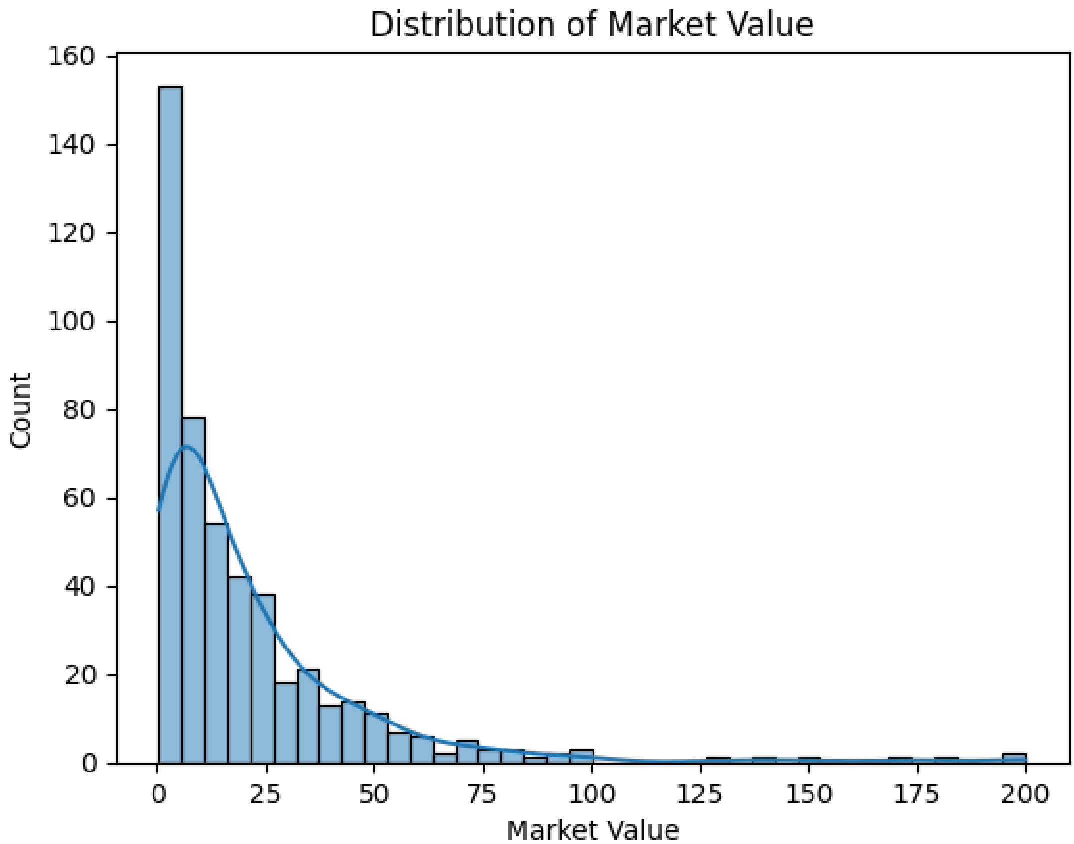
[13]:

```
sns.scatterplot(x='Goals-xG', y='Market Value', data=df)
plt.title('Overperformance vs Market Value')
plt.xlabel('Goals - xG')
plt.ylabel('Market Value')
plt.show()
```



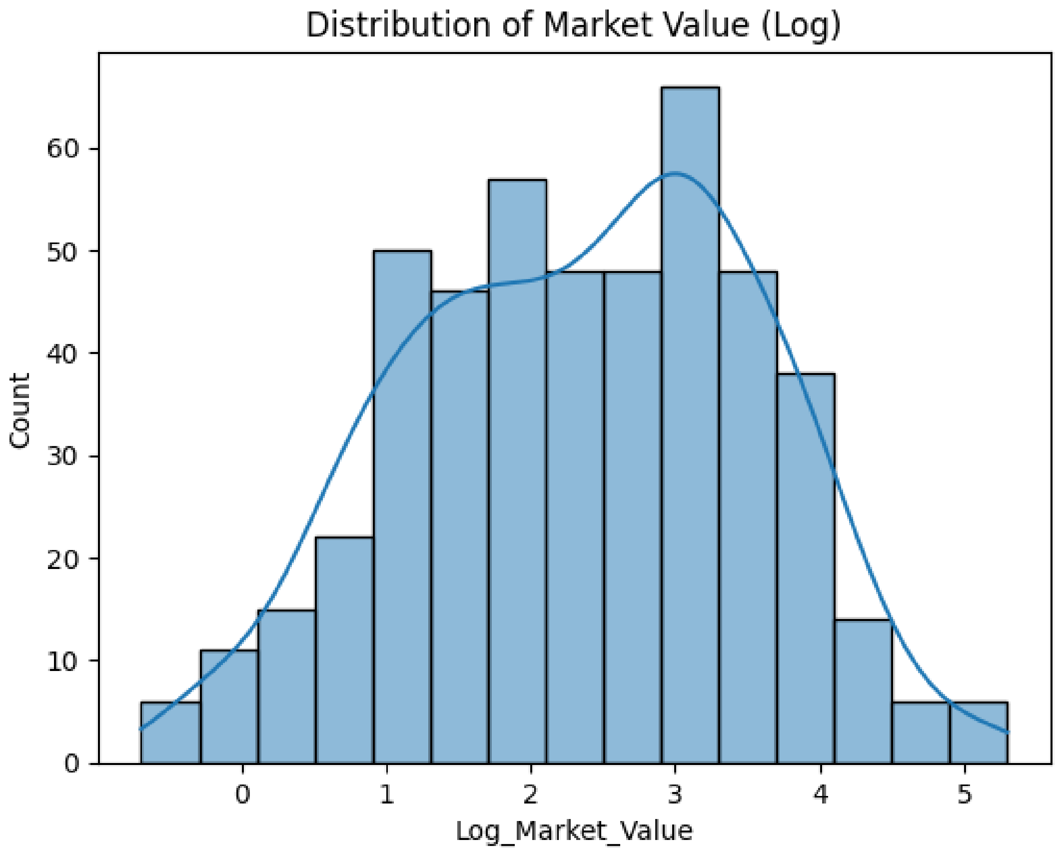
[14]:

```
sns.histplot(df['Market Value'], kde=True)
plt.title('Distribution of Market Value')
plt.show()
```



[15]:
```
df['Log_Market_Value'] = np.log(df['Market Value'])
```

[16]:
```
sns.histplot(df['Log_Market_Value'], kde=True, bins=15)
plt.title('Distribution of Market Value (Log)')
plt.show()
```



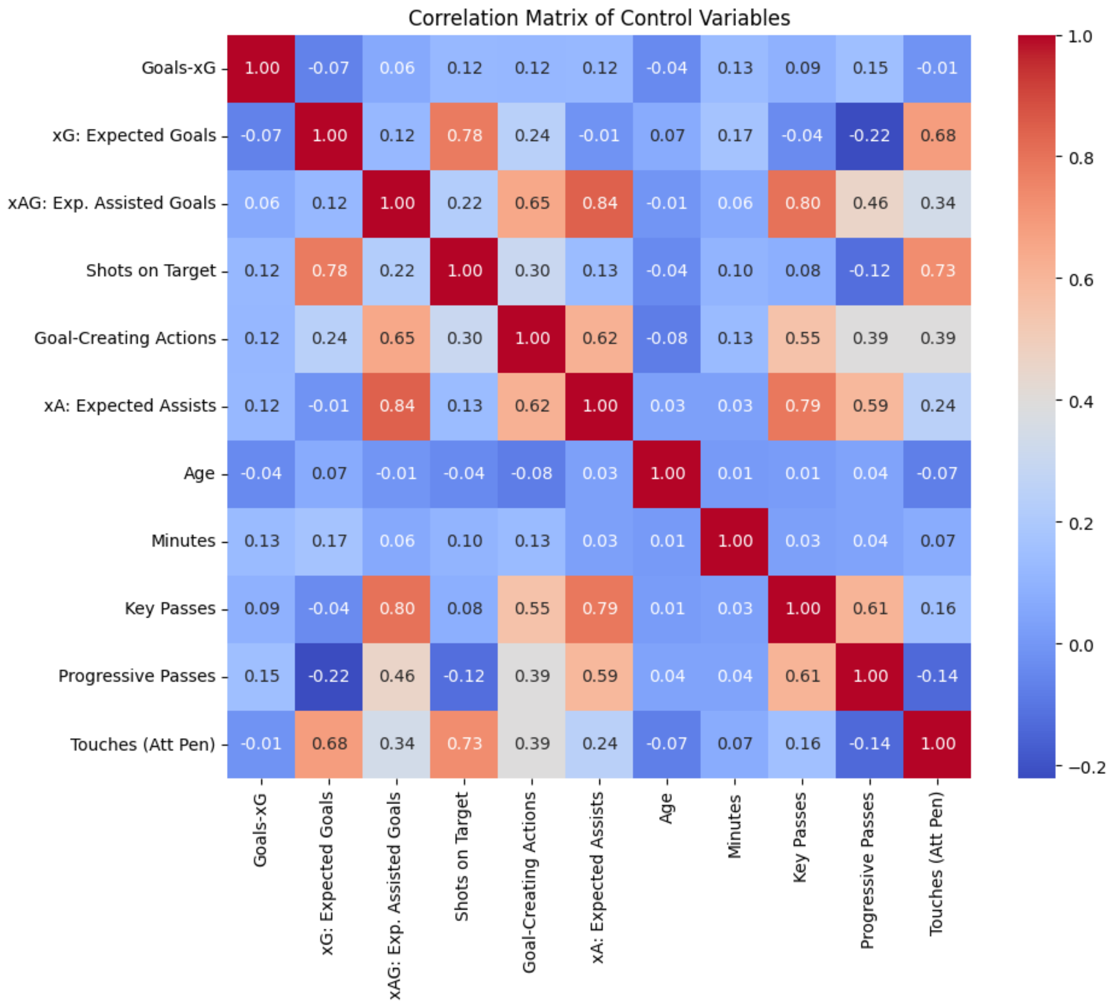
[17]:
```
df.columns
```

[17]:
```
Index(['Name', 'League', 'Goals', 'Assists', 'Non-Penalty Goals',
       'xG: Expected Goals', 'npxG: Non-Penalty xG',
       'xAG: Exp. Assisted Goals', 'Progressive Carries', 'Progressive Passes',
       'Progressive Passes Rec', 'Shots Total', 'Shots on Target',
       'Goals/Shot', 'npxG/Shot', 'xA: Expected Assists', 'Key Passes',
       'Through Balls', 'Crosses', 'Shot-Creating Actions',
       'Goal-Creating Actions', 'Shots on Target.1', 'Touches (Att 3rd)',
       'Touches (Att Pen)', 'Age', 'Market Value', 'Starting Eleven',
       'Minutes', 'Goal Involvement', 'Goals-xG', 'League_Strength',
       'Log_Market_Value'],
      dtype='object')
```

[18]:
```
features = ['Goals-xG','xG: Expected Goals', 'xAG: Exp. Assisted Goals', 'Shots␣
↪on Target', 'Goal-Creating Actions', 'xA: Expected Assists',
                 'Age', 'Minutes', 'Key Passes', 'Progressive Passes', 'Touches␣
↪(Att Pen)']

corr_matrix = df[features].corr()
```

```python
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Matrix of Control Variables')
plt.show()
```



0.0.3 3. Interaction Terms (Leave For Now)

[19]:
```python
# df['Age_X_League'] = df['League_Strength'] * df['Age']
```

[ ]:
```python

```

0.0.4 3. Dummy PL Variable

[58]:
```python
df['isPl'] = (df['League'] == 'epl').astype(int)
```

[26]:
```python
df['Pl_X_League'] = df['isPl'] * df['Age']
```

0.0.5 4. Prediction

[20]:
```python
df.isnull().values.any()
```

[20]:
```
False
```

[21]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 481 entries, 0 to 700
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Name                      481 non-null    object
 1   League                    481 non-null    object
 2   Goals                     481 non-null    float64
 3   Assists                   481 non-null    float64
 4   Non-Penalty Goals         481 non-null    float64
 5   xG: Expected Goals        481 non-null    float64
 6   npxG: Non-Penalty xG      481 non-null    float64
 7   xAG: Exp. Assisted Goals  481 non-null    float64
 8   Progressive Carries       481 non-null    float64
 9   Progressive Passes        481 non-null    float64
 10  Progressive Passes Rec    481 non-null    float64
 11  Shots Total               481 non-null    float64
 12  Shots on Target           481 non-null    float64
 13  Goals/Shot                481 non-null    float64
 14  npxG/Shot                 481 non-null    float64
 15  xA: Expected Assists      481 non-null    float64
 16  Key Passes                481 non-null    float64
 17  Through Balls             481 non-null    float64
 18  Crosses                   481 non-null    float64
 19  Shot-Creating Actions     481 non-null    float64
 20  Goal-Creating Actions     481 non-null    float64
 21  Shots on Target.1         481 non-null    float64
 22  Touches (Att 3rd)         481 non-null    float64
 23  Touches (Att Pen)         481 non-null    float64
 24  Age                       481 non-null    float64
 25  Market Value              481 non-null    float64
 26  Starting Eleven           481 non-null    float64
 27  Minutes                   481 non-null    float64
 28  Goal Involvement          481 non-null    float64
 29  Goals-xG                  481 non-null    float64
 30  League_Strength           481 non-null    float64
 31  Log_Market_Value          481 non-null    float64
```

```
dtypes: float64(30), object(2)
memory usage: 124.0+ KB
```

[61]:
```python
X = df[['Goals-xG','xG: Expected Goals','Goal-Creating Actions', 'Age',␣
↪'Minutes', 'League_Strength',
        'Progressive Passes', 'Touches (Att Pen)',
        # 'isPl',
        # 'Pl_X_League'
        ]]
y = df['Log_Market_Value']

X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log_Market_Value   R-squared:                       0.654
Model:                            OLS   Adj. R-squared:                  0.648
Method:                 Least Squares   F-statistic:                     111.3
Date:                Wed, 30 Apr 2025   Prob (F-statistic):         1.36e-103
Time:                        16:56:48   Log-Likelihood:                -518.72
No. Observations:                 481   AIC:                             1055.
Df Residuals:                     472   BIC:                             1093.
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
=========
                            coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
const                     2.6070      0.283      9.200      0.000       2.050
3.164
Goals-xG                  1.3379      0.311      4.308      0.000       0.728
1.948
xG: Expected Goals        1.9690      0.285      6.913      0.000       1.409
2.529
Goal-Creating Actions     1.1433      0.214      5.338      0.000       0.722
1.564
Age                      -0.1394      0.009    -15.485      0.000      -0.157
-0.122
Minutes                   0.0121      0.002      6.277      0.000       0.008
0.016
League_Strength           1.1299      0.106     10.650      0.000       0.921
1.338
Progressive Passes        0.1819      0.026      7.037      0.000       0.131
0.233
```

```
Touches (Att Pen)          0.1457      0.030      4.820      0.000       0.086
0.205
==============================================================================
Omnibus:                        0.479   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.304
Skew:                           0.012   Prob(JB):                        0.859
Kurtosis:                       3.121   Cond. No.                         643.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

0.0.6 VIF Test for Multicollinearity

[63]:
```python
X = df[['Goals-xG','xG: Expected Goals','Goal-Creating Actions', 'Age',␣
↪'Minutes', 'League_Strength',
        'Progressive Passes', 'Touches (Att Pen)',
        # 'isPl',
        # 'Pl_X_League'
        ]]
X = sm.add_constant(X)

vif_data = pd.DataFrame()
vif_data['feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.
↪shape[1])]

print(vif_data)
```

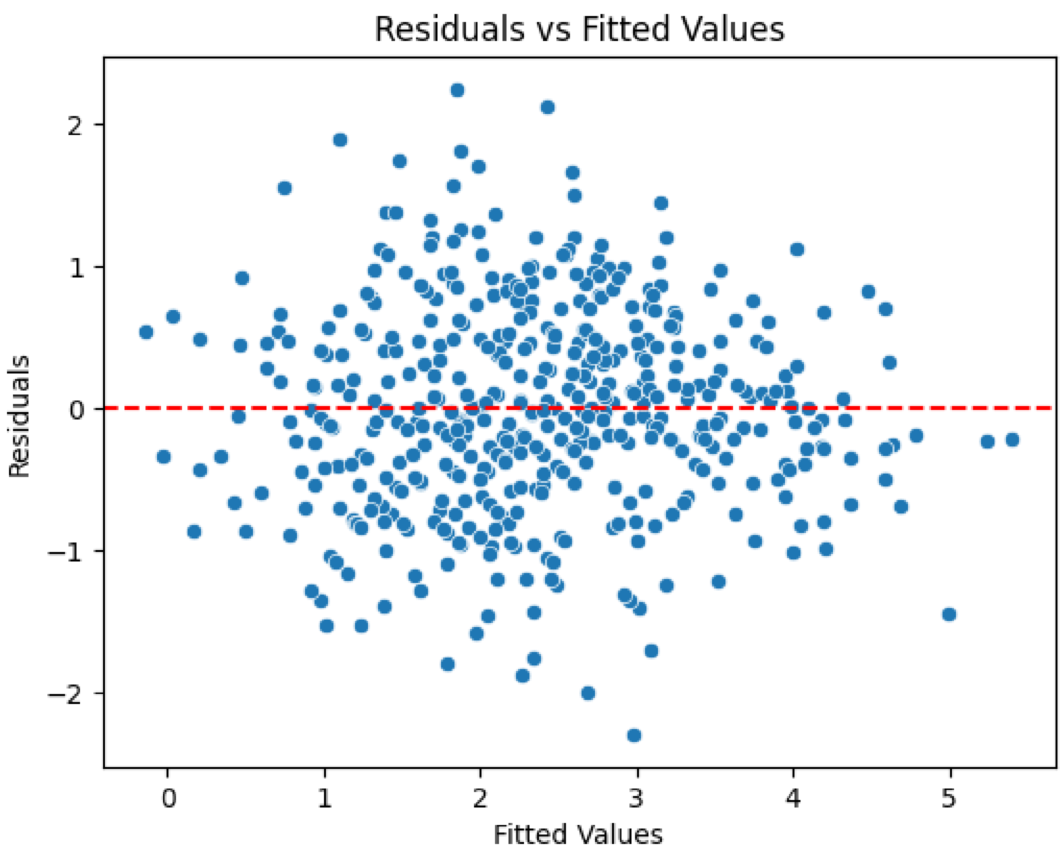
```
                 feature        VIF
0                  const  74.895568
1               Goals-xG   1.050186
2     xG: Expected Goals   2.094382
3  Goal-Creating Actions   1.593653
4                    Age   1.048913
5                Minutes   1.067950
6        League_Strength   1.021034
7     Progressive Passes   1.410793
8      Touches (Att Pen)   2.233812
```

[ ]:

0.0.7 Residual Plot and Heteroskedasticity

[64]:
```
residuals = model.resid
fitted = model.fittedvalues

sns.scatterplot(x=fitted, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values")
plt.show()
```



[65]:
```
from statsmodels.stats.diagnostic import het_breuschpagan

# y = dependent variable
# X = independent variables
from statsmodels.formula.api import ols
import statsmodels.api as sm

# if you've already run model = sm.OLS(y, X).fit()
```

```python
bp_test = het_breuschpagan(model.resid, model.model.exog)

labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, bp_test)))
```

```
{'LM Statistic': 15.41887519288316, 'LM-Test p-value': 0.05149451216662224,
'F-Statistic': 1.9539315232269931, 'F-Test p-value': 0.05053300224092823}
```

[66]:
```python
from statsmodels.stats.diagnostic import het_white

white_test = het_white(model.resid, model.model.exog)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, white_test)))
```

```
{'LM Statistic': 61.241628527944826, 'LM-Test p-value': 0.04360661647508923,
'F-Statistic': 1.4457099744693975, 'F-Test p-value': 0.03669506292616484}
```

0.0.8 Using a robust model

[ ]:
```python
# robust_model = model.get_robustcov_results(cov_type='HC3')
# print(robust_model.summary())

robust_model = sm.OLS(y, X).fit(cov_type='HC3')
print(robust_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log_Market_Value   R-squared:                       0.654
Model:                            OLS   Adj. R-squared:                  0.648
Method:                 Least Squares   F-statistic:                     141.3
Date:                Wed, 30 Apr 2025   Prob (F-statistic):          4.14e-120
Time:                        17:57:20   Log-Likelihood:                -518.72
No. Observations:                 481   AIC:                             1055.
Df Residuals:                     472   BIC:                             1093.
Df Model:                           8
Covariance Type:                  HC3
================================================================================
=========
                            coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
const                     2.6070      0.289      9.009      0.000       2.040
3.174
Goals-xG                  1.3379      0.313      4.272      0.000       0.724
1.952
xG: Expected Goals        1.9690      0.304      6.480      0.000       1.373
2.565
```

```
Goal-Creating Actions     1.1433      0.225      5.083      0.000       0.703
1.584
Age                      -0.1394      0.009    -15.610      0.000      -0.157
-0.122
Minutes                   0.0121      0.002      6.043      0.000       0.008
0.016
League_Strength           1.1299      0.098     11.476      0.000       0.937
1.323
Progressive Passes        0.1819      0.028      6.587      0.000       0.128
0.236
Touches (Att Pen)         0.1457      0.034      4.338      0.000       0.080
0.211
==============================================================================
Omnibus:                        0.479   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.304
Skew:                           0.012   Prob(JB):                        0.859
Kurtosis:                       3.121   Cond. No.                         643.
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC3)
```

[68]:

```python
bp_test = het_breuschpagan(robust_model.resid, robust_model.model.exog)

labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, bp_test)))
```

```
{'LM Statistic': 15.41887519288316, 'LM-Test p-value': 0.05149451216662224,
'F-Statistic': 1.9539315232269931, 'F-Test p-value': 0.05053300224092823}
```

[47]:

```python
white_test = het_white(robust_model.resid, robust_model.model.exog)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, white_test)))
```

```
{'LM Statistic': 61.241628527944826, 'LM-Test p-value': 0.043606661647508923,
'F-Statistic': 1.4457099744693975, 'F-Test p-value': 0.03669506292616484}
```

[ ]:

```python

```

0.0.9 Weighted Least Squares (WLS)

[70]:

```python
residuals = model.resid
weights = 1 / (residuals**2 + 1e-8)
# fitted = model.fittedvalues
# weights = 1 / (fitted**2)
wls_model = sm.WLS(y, X, weights=weights).fit()
print(wls_model.summary())
```

WLS Regression Results

Dep. Variable:	Log_Market_Value	R-squared:	1.000
Model:	WLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	4.108e+05
Date:	Wed, 30 Apr 2025	Prob (F-statistic):	0.00
Time:	18:08:39	Log-Likelihood:	-189.61
No. Observations:	481	AIC:	397.2
Df Residuals:	472	BIC:	434.8
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.6118	0.012	220.724	0.000	2.589	2.635
Goals-xG	1.3202	0.019	70.073	0.000	1.283	1.357
xG: Expected Goals	1.9772	0.024	81.862	0.000	1.930	2.025
Goal-Creating Actions	1.1408	0.009	129.576	0.000	1.124	1.158
Age	-0.1400	0.001	-241.401	0.000	-0.141	-0.139
Minutes	0.0122	0.000	93.951	0.000	0.012	0.012
League_Strength	1.1331	0.008	147.413	0.000	1.118	1.148
Progressive Passes	0.1829	0.001	133.936	0.000	0.180	0.186
Touches (Att Pen)	0.1454	0.002	76.460	0.000	0.142	0.149

Omnibus:	2033.546	Durbin-Watson:	1.892
Prob(Omnibus):	0.000	Jarque-Bera (JB):	79.000
Skew:	0.079	Prob(JB):	7.00e-18
Kurtosis:	1.021	Cond. No.	3.62e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.62e+03. This might indicate that there are strong multicollinearity or other numerical problems.

[49]:
```
bp_test = het_breuschpagan(wls_model.resid, wls_model.model.exog)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, bp_test)))
```

```
{'LM Statistic': 15.269812454904118, 'LM-Test p-value': 0.05410663173617769,
'F-Statistic': 1.9344224594677115, 'F-Test p-value': 0.05315302481065352}
```

[50]:
```
white_test = het_white(wls_model.resid, wls_model.model.exog)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, white_test)))
```

```
{'LM Statistic': 61.0638477565049, 'LM-Test p-value': 0.04503379229116959,
'F-Statistic': 1.4409028978463407, 'F-Test p-value': 0.0380324088033134}
```

[ ]:
```

```