ECON 323 Project

Bundesliga Match Winner Prediction using Python and Machine Learning

By Tammy Goel (50368463)

1. Introduction

1.1 Background

The Bundesliga is one of the most popular and competitive professional football leagues in Germany. With a rich history and passionate fanbase, analyzing and predicting outcomes in the Bundesliga can provide valuable insights for fans, bettors, and teams. Utilizing machine learning models to predict match winners and analyze factors influencing outcomes can be both informative and exciting.

1.2 Research Question

Can machine learning models accurately predict the winners of Bundesliga matches based on historical match data, team statistics, and match performance?

1.3 General Strategy

To answer this research question, we will employ various machine learning algorithms, such as random forests, regression or other suitable models. We will utilize a comprehensive dataset containing historical Bundesliga match data, including team attributes, match results, and other relevant variables. By training and evaluating different models, we aim to identify the most effective approach for predicting Bundesliga match winners.

1.4 Dataset

The dataset for this project will be sourced from reliable football databases, official Bundesliga records, or publicly available datasets specific to the Bundesliga. The dataset will include information such as match outcomes, team statistics (e.g., goals scored, possession, shots on target),and contextual factors (e.g., home/away advantage, referee decisions). The data will cover a significant period to ensure an adequate sample size for analysis.

The dataset will be preprocessed, ensuring data cleanliness, handling missing values, and appropriate feature engineering to extract relevant information for model training and evaluation.

By developing a predictive model using machine learning techniques, we aim to enhance our understanding of the factors influencing match outcomes in the Bundesliga and provide insights into the effectiveness of different algorithms for predicting winners. The results obtained from this project can contribute to the field of sports analytics and assist football enthusiasts in making informed predictions.

2. Cleaning Data amnd Exploration

In [345…
```
# Load all packages first
import pandas as pd
import re
import numpy as np
import folium
import branca
import branca.colormap as cm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_validate
from sklearn import linear_model
import seaborn as sns
from IPython.core.display import display, HTML
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from scipy import stats
from sklearn.neighbors import KNeighborsRegressor
pd.options.mode.chained_assignment = None
```

```
/tmp/ipykernel_160/3960108677.py:12: DeprecationWarning: Importing display
from IPython.core.display is deprecated since IPython 7.14, please import f
rom IPython display
  from IPython.core.display import display, HTML
```

In [346…
```
bundesliga = pd.read_csv("bundesliga.csv", index_col=0)
```

In [347…
```
bundesliga.shape
```

Out[347]: (1389, 27)

In [348…
```
bundesliga.head()
```

Out[348]:

	date	time	comp	round	day	venue	result	gf	ga	opponent	...	match report
1	2021-08-15	16:30	Bundesliga	Matchweek 1	Sun	Away	L	0.0	1.0	Bayer 04 Leverkusen	...	Match Report
2	2021-08-21	15:00	Bundesliga	Matchweek 2	Sat	Home	W	5.0	0.0	Holstein Kiel	...	Match Report
3	2021-08-28	12:30	Bundesliga	Matchweek 3	Sat	Home	W	5.0	0.0	RB Leipzig	...	Match Report
4	2021-09-11	15:00	Bundesliga	Matchweek 4	Sat	Away	W	1.0	0.0	VfB Stuttgart	...	Match Report
6	2021-09-18	15:00	Bundesliga	Matchweek 5	Sat	Home	D	0.0	0.0	FC Koln	...	Match Report

5 rows × 27 columns

In [349…
```
bundesliga["team"].value_counts()
```

Out[349]:
```
FC Koln                    72
SC Freiburg                72
FC Bayern Munich           72
Borussia Mönchengladbach   72
Hertha BSC                 72
FC Union Berlin            71
VfL Wolfsburg              71
TSG 1899 Hoffenheim        71
Borussia Dortmund          71
Eintracht Frankfurt        71
Bayer 04 Leverkusen        71
RB Leipzig                 71
VfB Stuttgart              70
FC Nurnberg                70
FC Schalke 04              70
FSV Mainz 05               70
VfL Bochum                 38
FC Augsburg                38
Arminia Bielefeld          38
SC Paderborn               38
Fortuna Düsseldorf         34
SV Darmstadt 98            33
Holstein Kiel              33
Name: team, dtype: int64
```

In [350…
```
bundesliga[bundesliga["team"] == "VfL Bochum"].sort_values("date")
```

Out[350]:

	date	time	comp	round	day	venue	result	gf	ga	opponent	.
1	2020-09-12	17:30	Bundesliga	Matchweek 1	Sat	Home	W	4.0	3.0	VfL Wolfsburg	.
2	2020-09-20	16:30	Bundesliga	Matchweek 2	Sun	Away	W	2.0	0.0	FC Nurnberg	.
4	2020-09-28	20:00	Bundesliga	Matchweek 3	Mon	Home	W	3.0	1.0	RB Leipzig	.
6	2020-10-04	19:15	Bundesliga	Matchweek 4	Sun	Away	L	2.0	7.0	FC Schalke 04	.
7	2020-10-17	12:30	Bundesliga	Matchweek 5	Sat	Away	D	2.0	2.0	FSV Mainz 05	.
9	2020-10-24	20:00	Bundesliga	Matchweek 6	Sat	Home	W	2.0	1.0	SC Paderborn	.
11	2020-10-31	17:30	Bundesliga	Matchweek 7	Sat	Home	W	2.0	1.0	Borussia Mönchengladbach	.
13	2020-11-08	16:30	Bundesliga	Matchweek 8	Sun	Away	D	1.0	1.0	Borussia Dortmund	.
14	2020-11-22	19:15	Bundesliga	Matchweek 9	Sun	Home	W	3.0	0.0	VfB Stuttgart	.
16	2020-11-28	12:30	Bundesliga	Matchweek 10	Sat	Away	D	1.0	1.0	SC Freiburg	.
18	2020-12-06	19:15	Bundesliga	Matchweek 11	Sun	Home	W	4.0	0.0	Eintracht Frankfurt	.
20	2020-12-13	16:30	Bundesliga	Matchweek 12	Sun	Away	D	1.0	1.0	FC Augsburg	.
21	2020-12-16	20:00	Bundesliga	Matchweek 13	Wed	Home	W	2.0	1.0	Bayer 04 Leverkusen	.
22	2020-12-19	12:30	Bundesliga	Matchweek 14	Sat	Away	W	7.0	0.0	TSG 1899 Hoffenheim	.
23	2020-12-27	16:30	Bundesliga	Matchweek 15	Sun	Home	D	1.0	1.0	Arminia Bielefeld	.
24	2020-12-30	20:00	Bundesliga	Matchweek 16	Wed	Away	D	0.0	0.0	Hertha BSC	.
25	2021-01-04	20:00	Bundesliga	Matchweek 17	Mon	Away	L	0.0	1.0	FC Koln	.
27	2021-01-17	16:30	Bundesliga	Matchweek 19	Sun	Home	D	0.0	0.0	FC Bayern Munich	.
28	2021-01-21	20:00	Bundesliga	Matchweek 18	Thu	Home	L	0.0	1.0	FC Union Berlin	.
30	2021-01-28	20:00	Bundesliga	Matchweek 20	Thu	Away	W	3.0	1.0	Bayer 04 Leverkusen	.
31	2021-01-31	16:30	Bundesliga	Matchweek 21	Sun	Away	W	3.0	1.0	Borussia Mönchengladbach	.

	date	time	comp	round	day	venue	result	gf	ga	opponent	.
32	2021-02-03	20:15	Bundesliga	Matchweek 22	Wed	Home	L	0.0	1.0	SC Freiburg	.
33	2021-02-07	16:30	Bundesliga	Matchweek 23	Sun	Home	L	1.0	4.0	Borussia Dortmund	.
34	2021-02-13	12:30	Bundesliga	Matchweek 24	Sat	Away	L	1.0	3.0	VfB Stuttgart	.
36	2021-02-20	17:30	Bundesliga	Matchweek 25	Sat	Home	L	0.0	2.0	FSV Mainz 05	.
37	2021-02-28	19:15	Bundesliga	Matchweek 26	Sun	Away	W	2.0	0.0	SC Paderborn	.
38	2021-03-04	20:15	Bundesliga	Matchweek 29	Thu	Home	L	0.0	1.0	FC Nurnberg	.
39	2021-03-07	14:00	Bundesliga	Matchweek 27	Sun	Home	L	0.0	1.0	FC Augsburg	.
41	2021-03-15	20:00	Bundesliga	Matchweek 28	Mon	Away	W	1.0	0.0	Eintracht Frankfurt	.
42	2021-04-03	20:00	Bundesliga	Matchweek 30	Sat	Away	W	3.0	0.0	RB Leipzig	.
44	2021-04-10	15:00	Bundesliga	Matchweek 31	Sat	Home	W	2.0	1.0	FC Schalke 04	.
46	2021-04-19	20:00	Bundesliga	Matchweek 32	Mon	Away	D	1.0	1.0	VfL Wolfsburg	.
47	2021-04-24	12:30	Bundesliga	Matchweek 33	Sat	Home	D	1.0	1.0	Hertha BSC	.
48	2021-05-08	20:15	Bundesliga	Matchweek 35	Sat	Home	W	2.0	0.0	FC Koln	.
49	2021-05-13	20:15	Bundesliga	Matchweek 34	Thu	Away	W	4.0	2.0	FC Bayern Munich	.
50	2021-05-16	16:30	Bundesliga	Matchweek 36	Sun	Away	W	2.0	1.0	Arminia Bielefeld	.
51	2021-05-19	20:15	Bundesliga	Matchweek 37	Wed	Away	W	3.0	0.0	FC Union Berlin	.
52	2021-05-23	16:00	Bundesliga	Matchweek 38	Sun	Home	W	2.0	0.0	TSG 1899 Hoffenheim	.

38 rows × 27 columns

In [351…
```
bundesliga.dtypes
```

Out[351]:
```
date             object
time             object
comp             object
round            object
day              object
venue            object
result           object
gf              float64
ga              float64
opponent         object
xg              float64
xga             float64
poss            float64
attendance      float64
captain          object
formation        object
referee          object
match report     object
notes           float64
sh              float64
sot             float64
dist            float64
fk              float64
pk              float64
pkatt           float64
season            int64
team             object
dtype: object
```

In [352…
```
bundesliga["round"].value_counts()
```

Out[352]:
```
Matchweek 1     39
Matchweek 16    39
Matchweek 34    39
Matchweek 32    39
Matchweek 31    39
Matchweek 29    39
Matchweek 28    39
Matchweek 26    39
Matchweek 25    39
Matchweek 24    39
Matchweek 23    39
Matchweek 2     39
Matchweek 19    39
Matchweek 17    39
Matchweek 20    39
Matchweek 15    39
Matchweek 5     39
Matchweek 3     39
Matchweek 13    39
Matchweek 12    39
Matchweek 4     39
Matchweek 11    39
Matchweek 10    39
Matchweek 9     39
Matchweek 8     39
Matchweek 14    39
Matchweek 7     39
Matchweek 6     39
Matchweek 30    37
Matchweek 27    37
Matchweek 22    37
Matchweek 21    37
Matchweek 18    37
Matchweek 33    32
Matchweek 35    20
Matchweek 36    20
Matchweek 37    20
Matchweek 38    20
Name: round, dtype: int64
```

In [353…
```
del bundesliga["comp"]
```

In [354…
```
del bundesliga["notes"]
```

Removing and transforming object types, By cleaning these object types, I want to enhance the quality and usefulness of my dataset for further analysis and modeling purposes.

In [355…
```
bundesliga["date"] = pd.to_datetime(bundesliga["date"])
```

Converting target to an integer for more usefulnesss, where loss or draw is a 0, winning is 1

In [356…
```
bundesliga["target"] = (bundesliga["result"] == "W").astype("int")
```

Converting some strings to numeric codes to leverage the usefulness of the numbers.

In [357…
```
bundesliga["venue_code"] = bundesliga["venue"].astype("category").cat.codes
```

In [358…
```
bundesliga["opp_code"] = bundesliga["opponent"].astype("category").cat.codes
```

Removing unnecessary characters from some column names

In [359…
```
bundesliga["hour"] = bundesliga["time"].str.replace(":.+", "", regex=True).a
```

In [360…
```
bundesliga["day_code"] = bundesliga["date"].dt.dayofweek
```

3. Modelling using Machine Learning

The opponent code variable in my project exhibits non-linear patterns, and utilizing a random forest algorithm will enable me to capture and identify these non-linearities effectively. By employing random forest, I can also uncover complex relationships and correlations between the opponent code and other variables in the dataset, leading to more accurate predictions and improved model performance.

In [361…
```
from sklearn.ensemble import RandomForestClassifier
```

In [362…
```
rf = RandomForestClassifier(n_estimators=50, min_samples_split=10, random_st
```

In [363…
```
# spliting training and testing data

train = bundesliga[bundesliga["date"] < '2022-01-01']
```

In [364…
```
test = bundesliga[bundesliga["date"] > '2022-01-01']
```

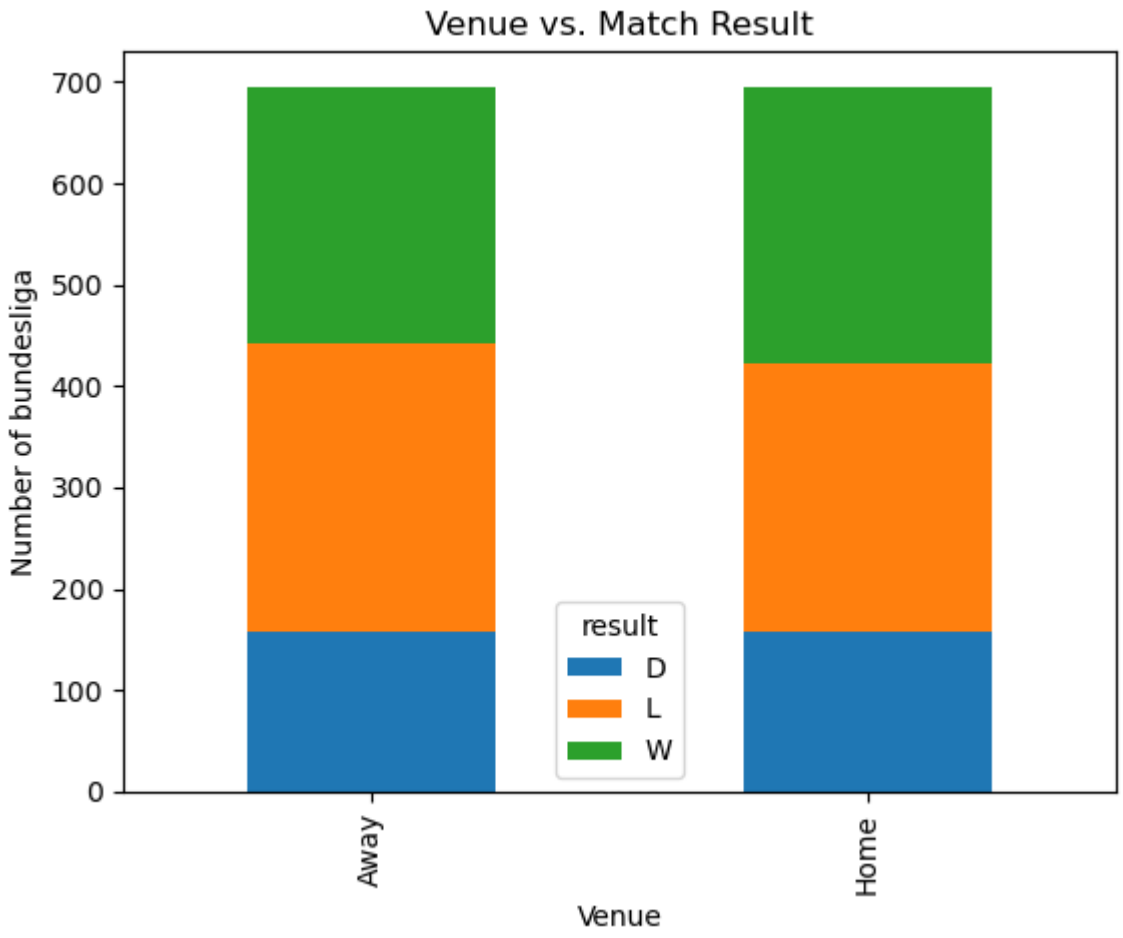
3.1 Visualizations

In this seciton, the aim is to identify the most influential variables for modeling by employing visualizations. The goal is to understand which factors have the highest impact on the outcome. By examining the relationships between variables and the target variable through visualizations, we can gain insights into the key factors that significantly affect the results. This analysis will help in selecting the most relevant variables for building accurate and effective models.

Venue vs The Result

In [365…
```
import seaborn as sns
import matplotlib.pyplot as plt

# Count the number of bundesliga by result and venue
```

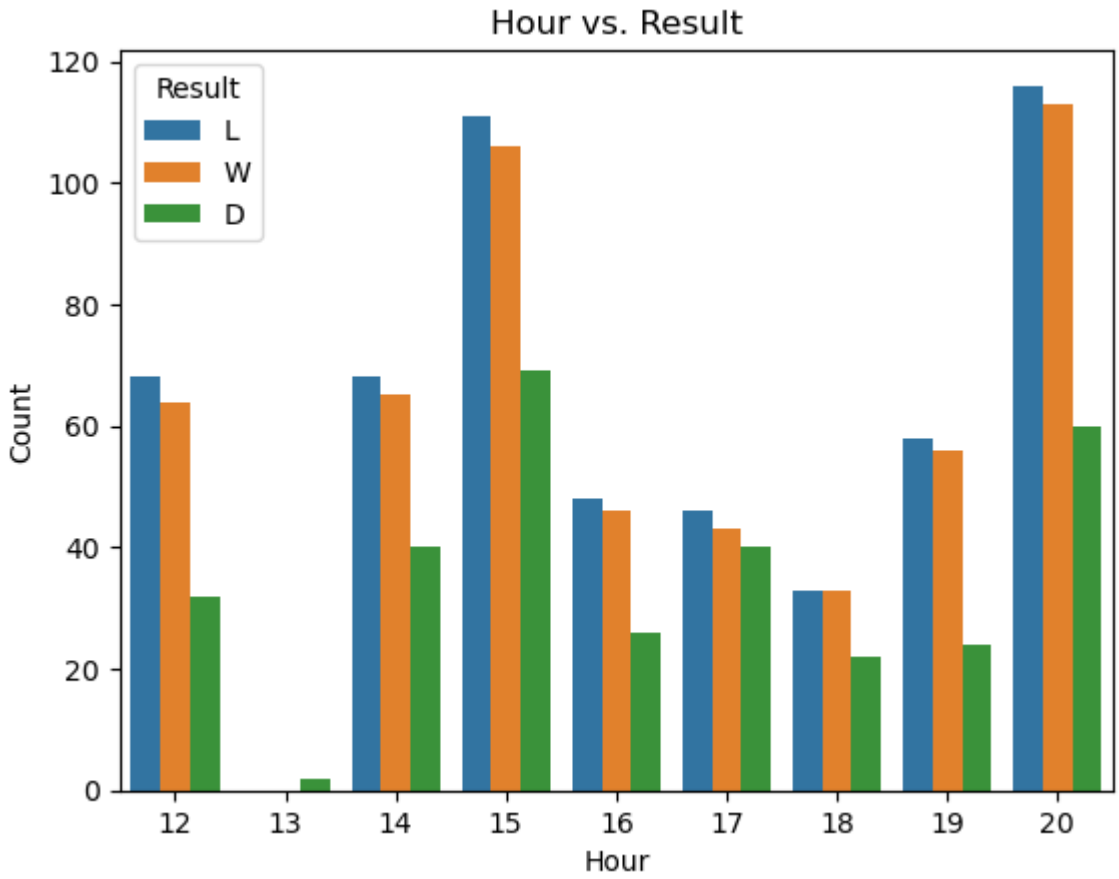
```
result_venue_counts = bundesliga.groupby(['venue', 'result']).size().unstack

# Plot the bar plot
result_venue_counts.plot(kind='bar', stacked=True)

# Set the labels and title
plt.xlabel('Venue')
plt.ylabel('Number of bundesliga')
plt.title('Venue vs. Match Result')

# Show the plot
plt.show()
```



Hour of the Day vs The Result

In [366…

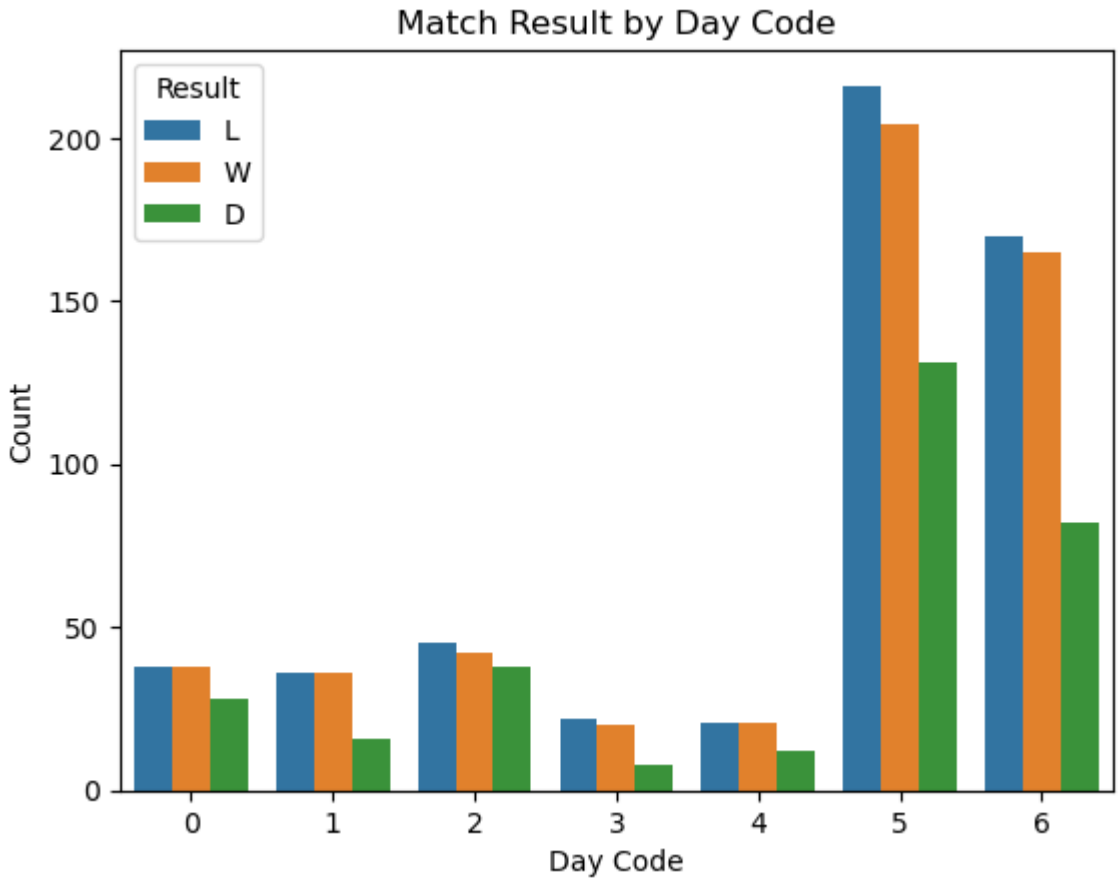
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a bar plot of hour vs. result
sns.countplot(x="hour", hue="result", data=bundesliga)
plt.xlabel("Hour")
plt.ylabel("Count")
plt.title("Hour vs. Result")
plt.legend(title="Result")
plt.show()
```



Day of the Month vs The Result

In [367…

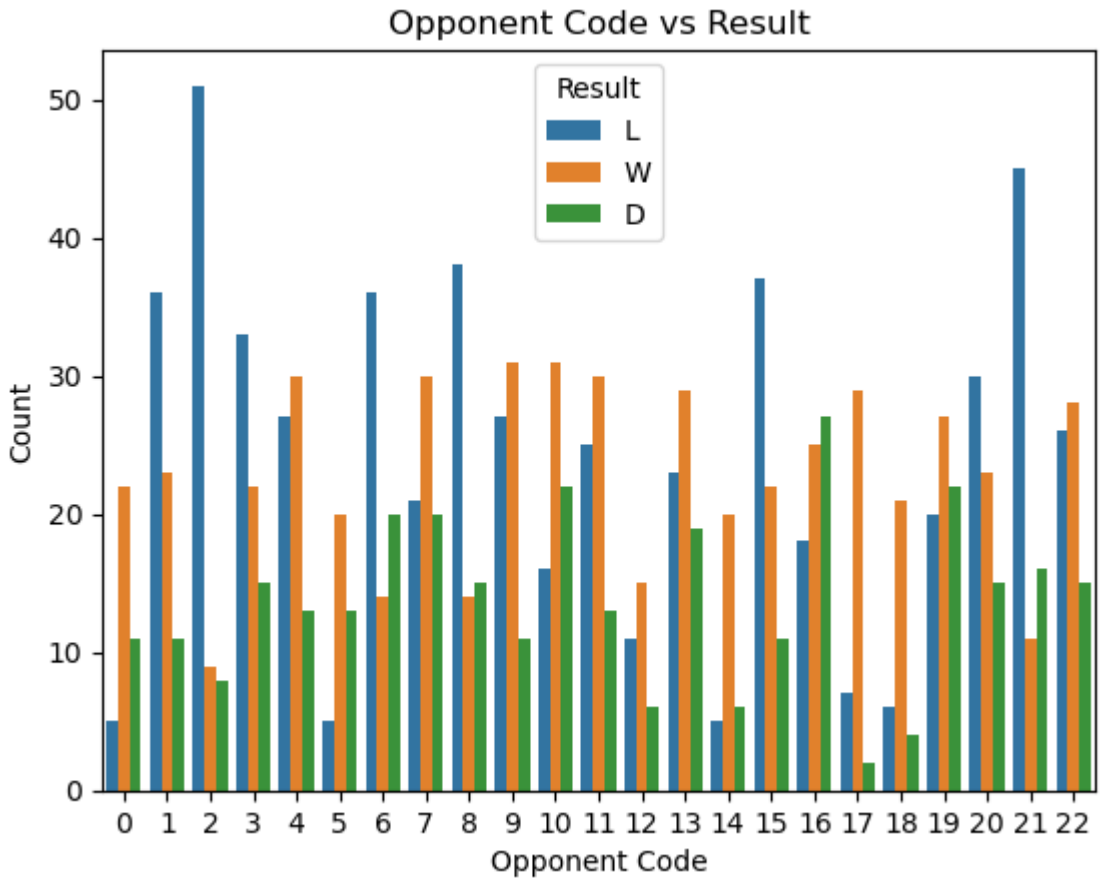
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x="day_code", hue="result", data=bundesliga)
plt.xlabel("Day Code")
plt.ylabel("Count")
plt.title("Match Result by Day Code")
plt.legend(title="Result")
plt.show()
```



Opponent Code vs The Result

In [368…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting opp_code vs result
sns.countplot(x="opp_code", hue="result", data=bundesliga)
plt.xlabel("Opponent Code")
plt.ylabel("Count")
plt.title("Opponent Code vs Result")
plt.legend(title="Result")
plt.show()
```



In [369…
```
team_name = bundesliga.loc[bundesliga["opp_code"] == 13, "team"].unique()[0]
print(team_name)
```

```
Borussia Dortmund
```

Opponent Code vs The Result

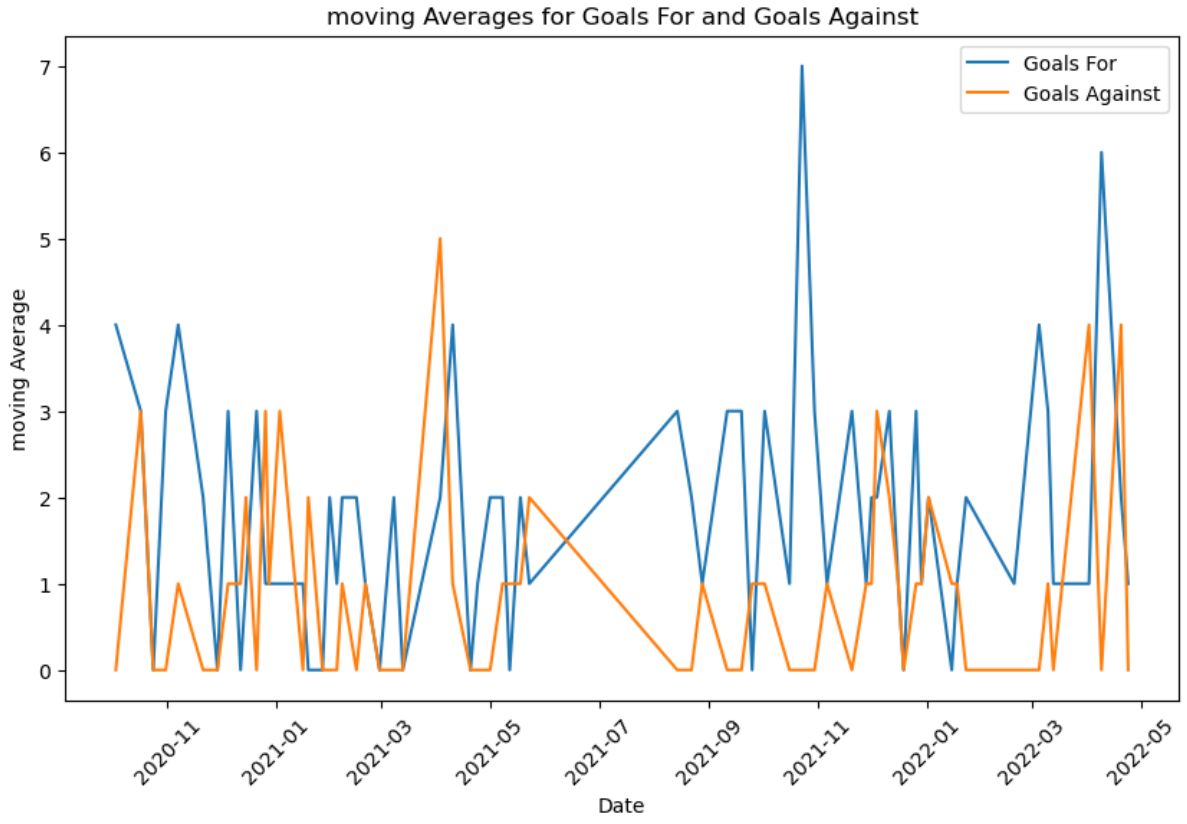
In [370…
```
team_name = bundesliga.loc[bundesliga["opp_code"] == 13, "team"].unique()[0]
print(team_name)
```

```
Borussia Dortmund
```

In [ ]:
```
```

In [371…
```
import matplotlib.pyplot as plt

#To create a visualization graphic for the moving averages method, you can u

# Select the data for a specific team
team_data = bundesliga_moving.loc[bundesliga_moving['team'] == "FC Nurnberg"

# Extract the dates and moving averages for goals for and goals against
dates = team_data['date']
gf_moving = team_data['gf']
ga_moving = team_data['ga']

# Plot the moving averages
plt.figure(figsize=(10, 6))
```

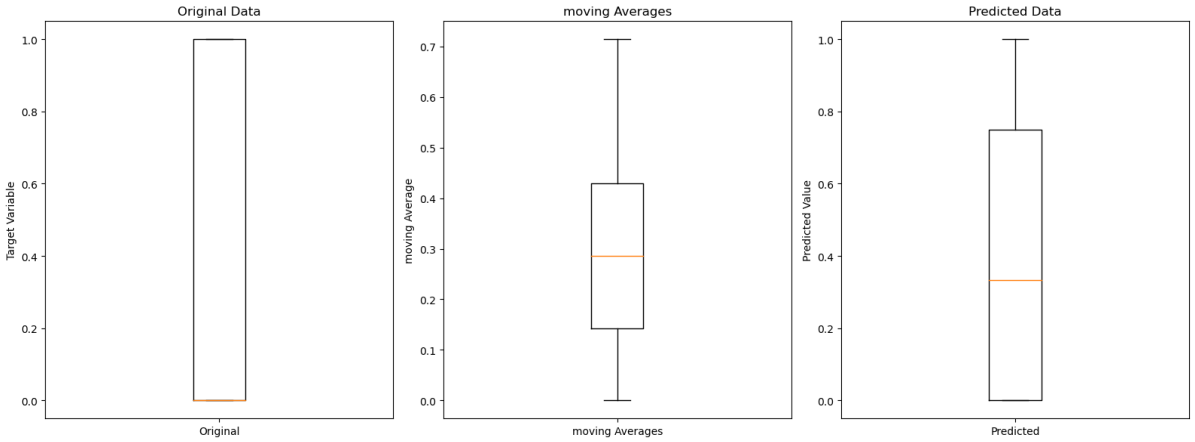
```
plt.plot(dates, gf_moving, label='Goals For')
plt.plot(dates, ga_moving, label='Goals Against')

# Customize the plot
plt.title('moving Averages for Goals For and Goals Against')
plt.xlabel('Date')
plt.ylabel('moving Average')
plt.xticks(rotation=45)
plt.legend()

# Display the plot
plt.show()
```



In [373…

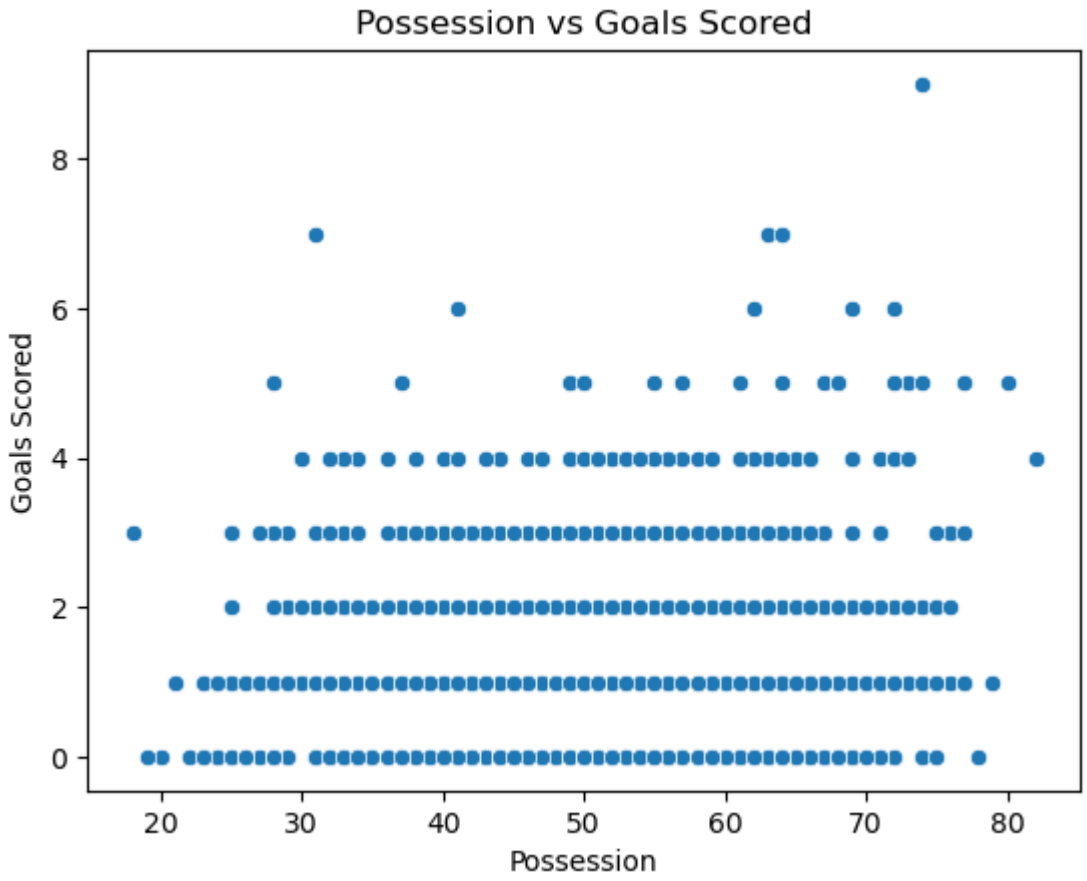
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have a DataFrame named 'bundesliga' with columns 'venue_code'

# moving Averages Method
moving_avg = bundesliga.groupby(["venue_code", "opp_code", "hour", "day_code
bundesliga.reset_index(drop=True, inplace=True)
bundesliga["moving_avg"] = moving_avg

# Using Predictors
predictors = ["venue_code", "opp_code", "hour", "day_code"]
predicted = bundesliga.groupby(predictors)["target"].mean()

# Create a Box Plot
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(16, 6))

# Box Plot for Original Data
```

```
axes[0].boxplot(bundesliga["target"])
axes[0].set_title("Original Data")
axes[0].set_ylabel("Target Variable")
axes[0].set_xticklabels(["Original"])  # Set x-axis labels for this subplot

# Box Plot for moving Averages
axes[1].boxplot(bundesliga["moving_avg"].dropna())
axes[1].set_title("moving Averages")
axes[1].set_ylabel("moving Average")
axes[1].set_xticklabels(["moving Averages"])  # Set x-axis labels for this s

# Box Plot for Predicted Data
axes[2].boxplot(predicted)
axes[2].set_title("Predicted Data")
axes[2].set_ylabel("Predicted Value")
axes[2].set_xticklabels(["Predicted"])  # Set x-axis labels for this subplot

plt.tight_layout()
plt.show()
```



In [374…

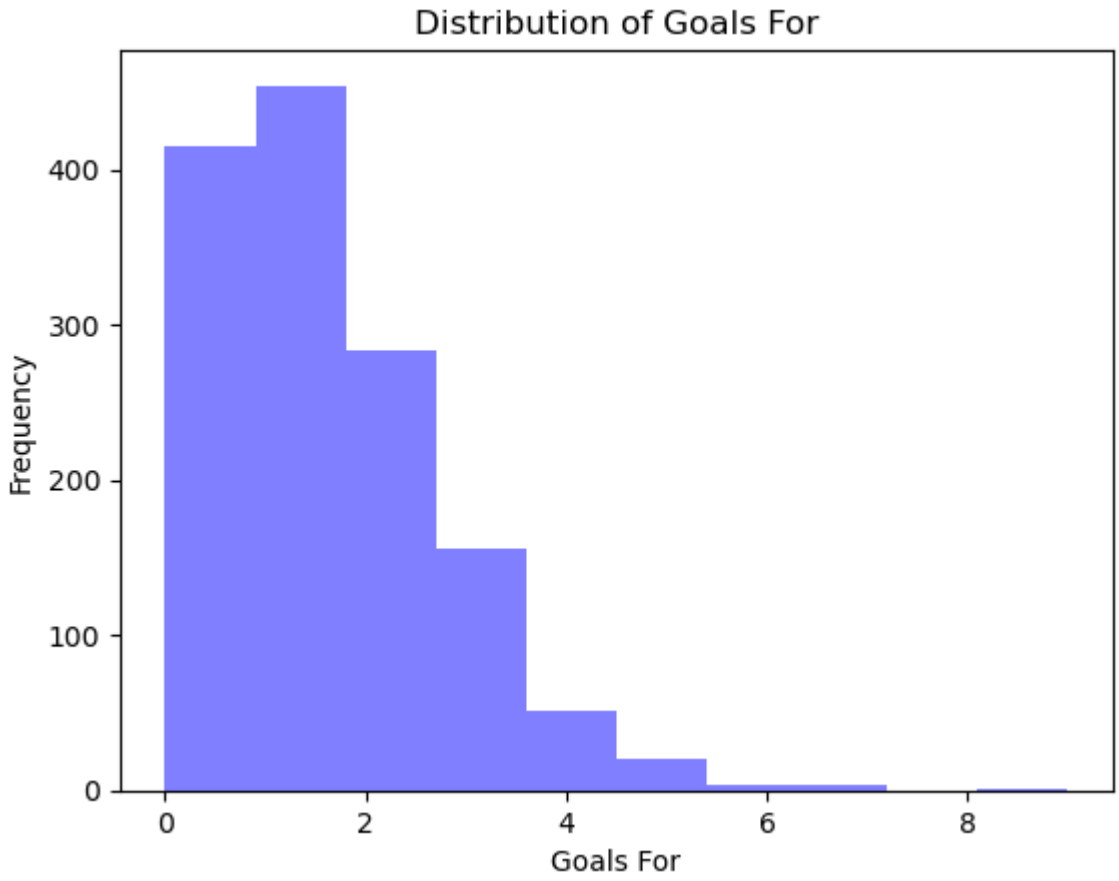
```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualize the relationship between possession and goals scored
sns.scatterplot(x='poss', y='gf', data=bundesliga)
plt.xlabel('Possession')
plt.ylabel('Goals Scored')
plt.title('Possession vs Goals Scored')
plt.show()
```

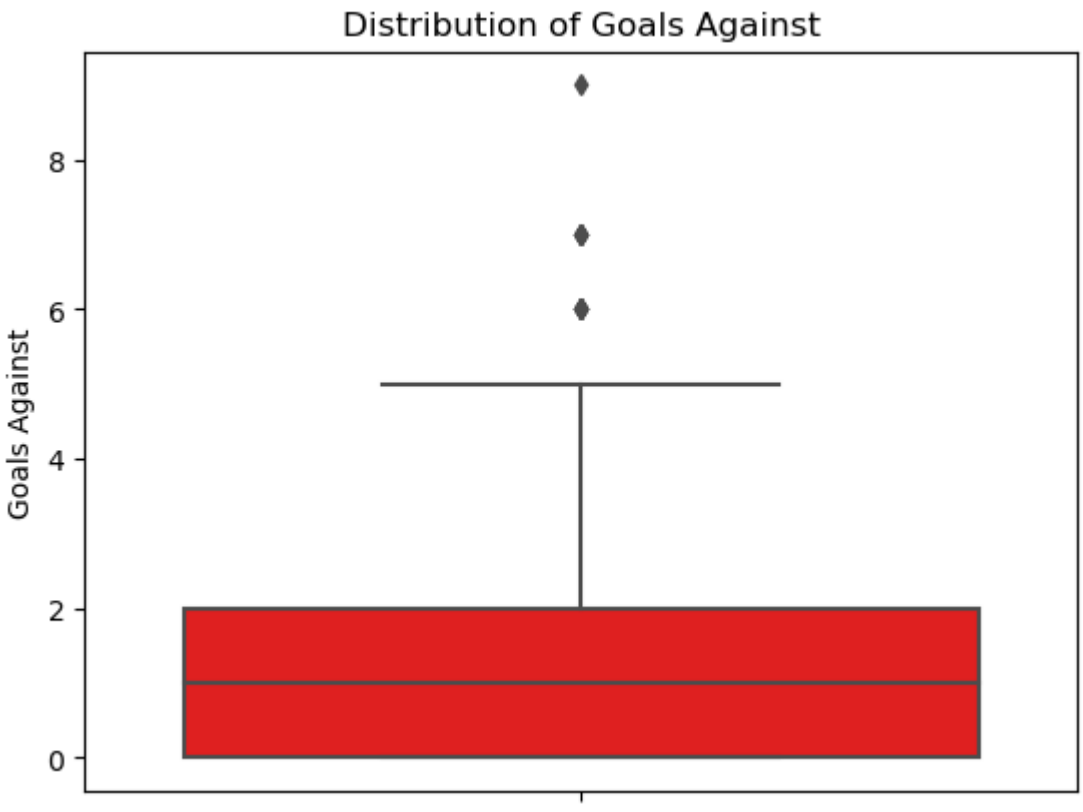


This graph shows no stroing relation between possession and scoring more goals which might be non-intiutive

In [375…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Extract goals for and goals against columns
goals_for = bundesliga['gf']
goals_against = bundesliga['ga']

# Plot histogram of goals for
plt.hist(goals_for, bins=10, color='blue', alpha=0.5)
plt.xlabel('Goals For')
plt.ylabel('Frequency')
plt.title('Distribution of Goals For')
plt.show()

# Plot boxplot of goals against
sns.boxplot(y=goals_against, color='red')
plt.ylabel('Goals Against')
plt.title('Distribution of Goals Against')
plt.show()
```





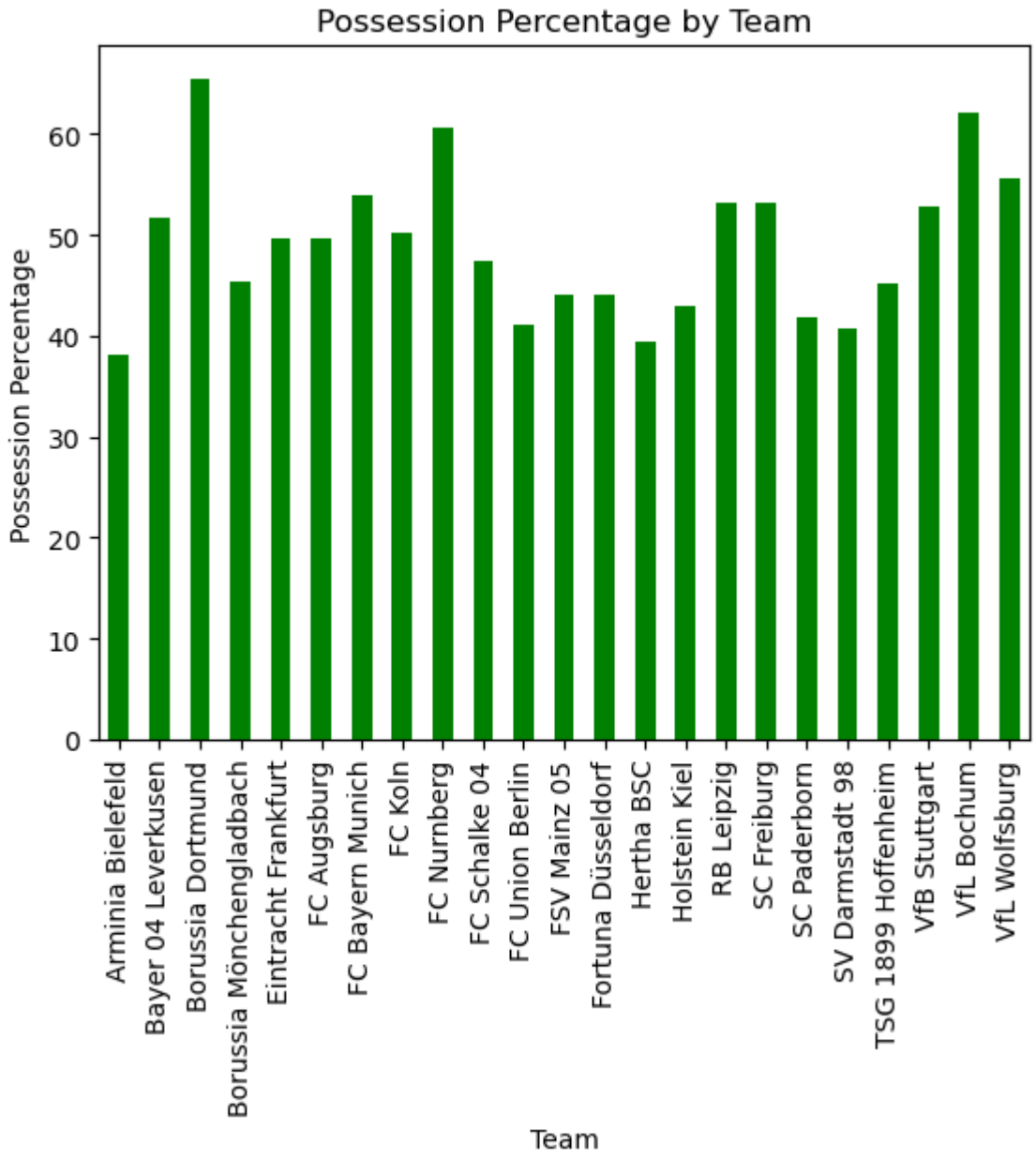
In [376…

```
actual_goals = bundesliga['gf']
expected_goals = bundesliga['xg']

# Calculate the difference between actual goals and expected goals
```

```
goal_difference = actual_goals - expected_goals

# Identify overperforming or underperforming teams
overperforming_teams = bundesliga[goal_difference > 0]
underperforming_teams = bundesliga[goal_difference < 0]
```

In [377…

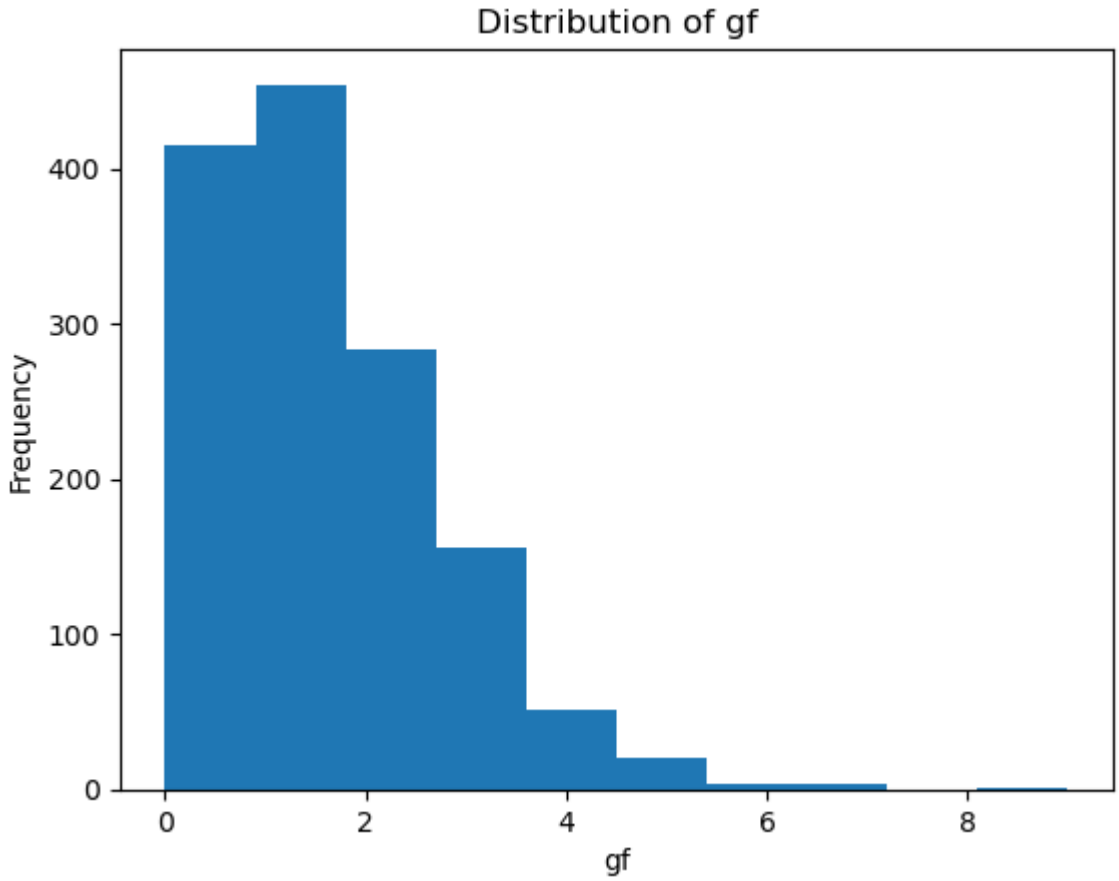
```
# C

possession = bundesliga['poss']

# Compare possession percentages of different teams
team_possession = bundesliga.groupby('team')['poss'].mean()

# Analyze possession trends over time
time_possession = bundesliga.groupby('date')['poss'].mean()

# Plot possession percentages of different teams
team_possession.plot(kind='bar', color='green')
plt.xlabel('Team')
plt.ylabel('Possession Percentage')
plt.title('Possession Percentage by Team')
plt.show()
```

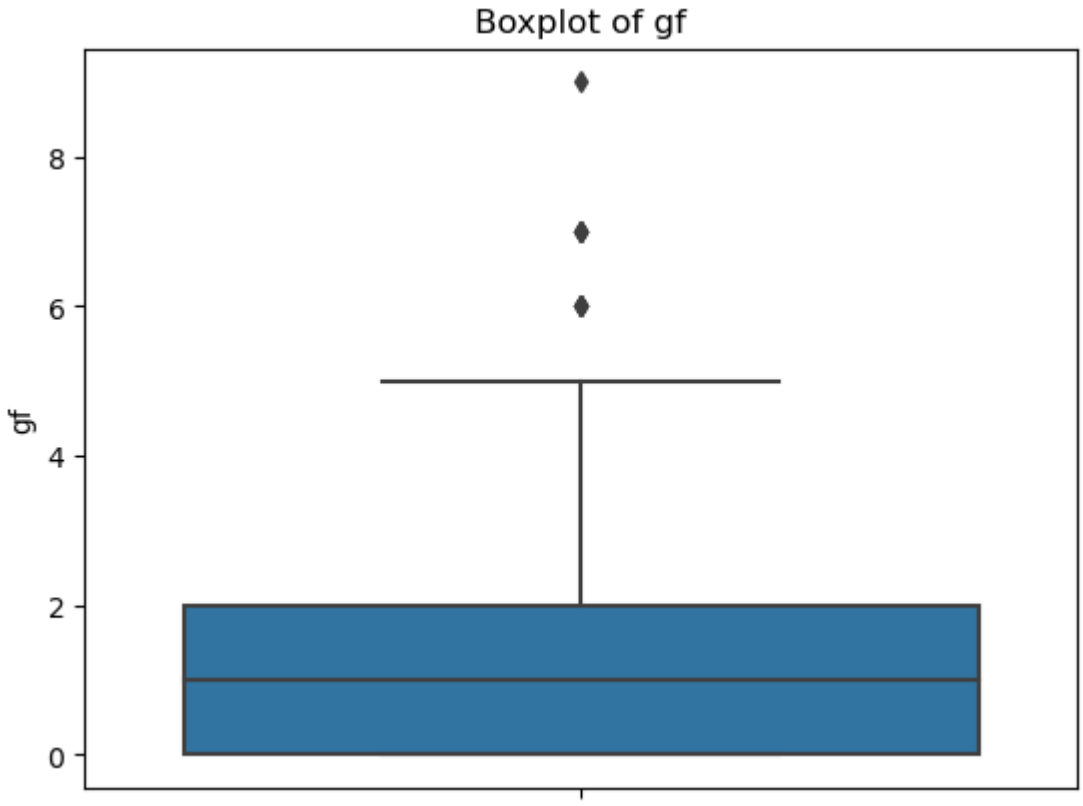


In [378…

```python
#Quantitative

import matplotlib.pyplot as plt

# Select a quantitative variable
quantitative_variable = 'gf'

# Plot a histogram of the quantitative variable
plt.hist(bundesliga[quantitative_variable], bins=10)
plt.xlabel(quantitative_variable)
plt.ylabel('Frequency')
plt.title('Distribution of {}'.format(quantitative_variable))
plt.show()

# Plot a boxplot of the quantitative variable
sns.boxplot(data=bundesliga, y=quantitative_variable)
plt.ylabel(quantitative_variable)
```

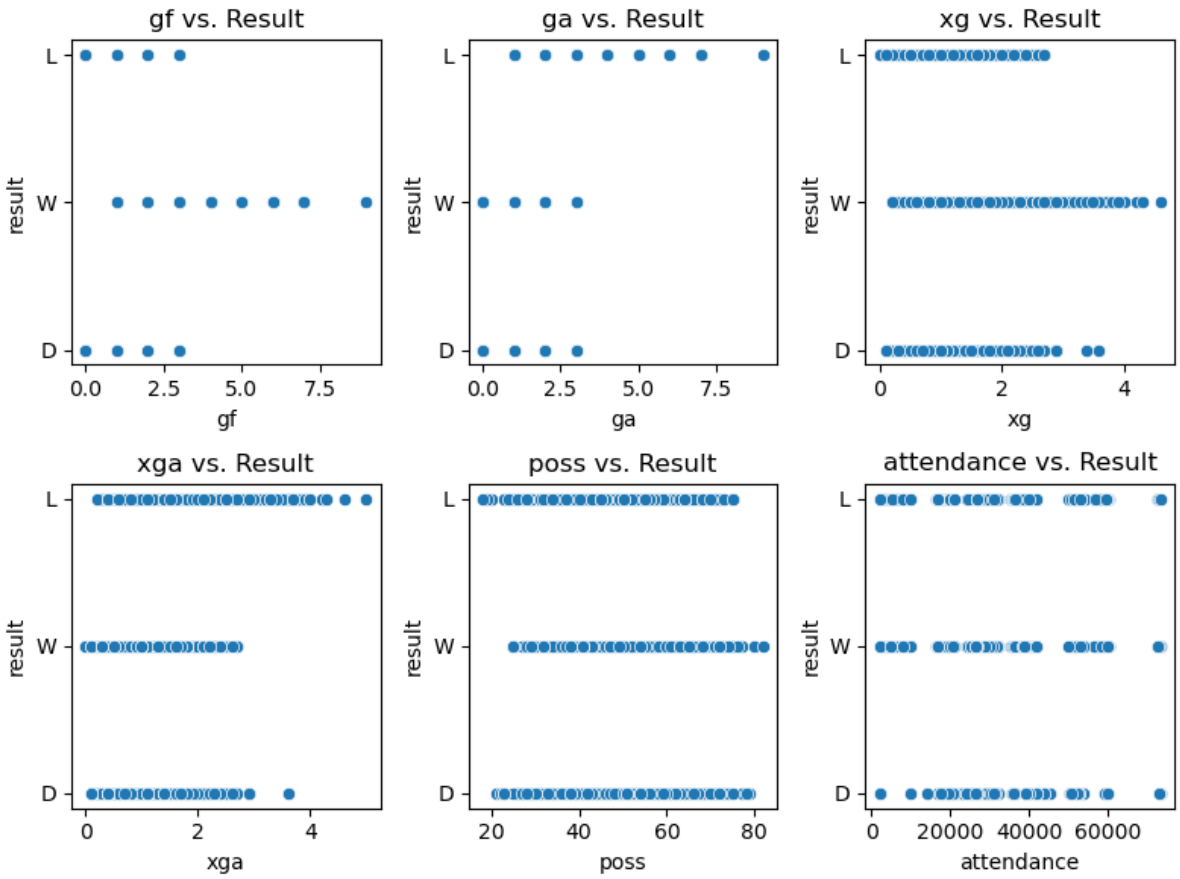
```
plt.title('Boxplot of {}'.format(quantitative_variable))
plt.show()
```





In [379…

```
fig, axes = plt.subplots(2, 3, figsize=(8, 6))
colnames = ['gf', 'ga', 'xg', 'xga', 'poss', 'attendance']

for i in range(len(colnames)):
    if i < 3:
        sns.scatterplot(x=colnames[i], y="result", data=bundesliga, ax=axes[
        axes[0, i].set_title(f"{colnames[i]} vs. Result")
    else:
        sns.scatterplot(x=colnames[i], y="result", data=bundesliga, ax=axes[
        axes[1, i-3].set_title(f"{colnames[i]} vs. Result")

fig.tight_layout()
plt.show()
```

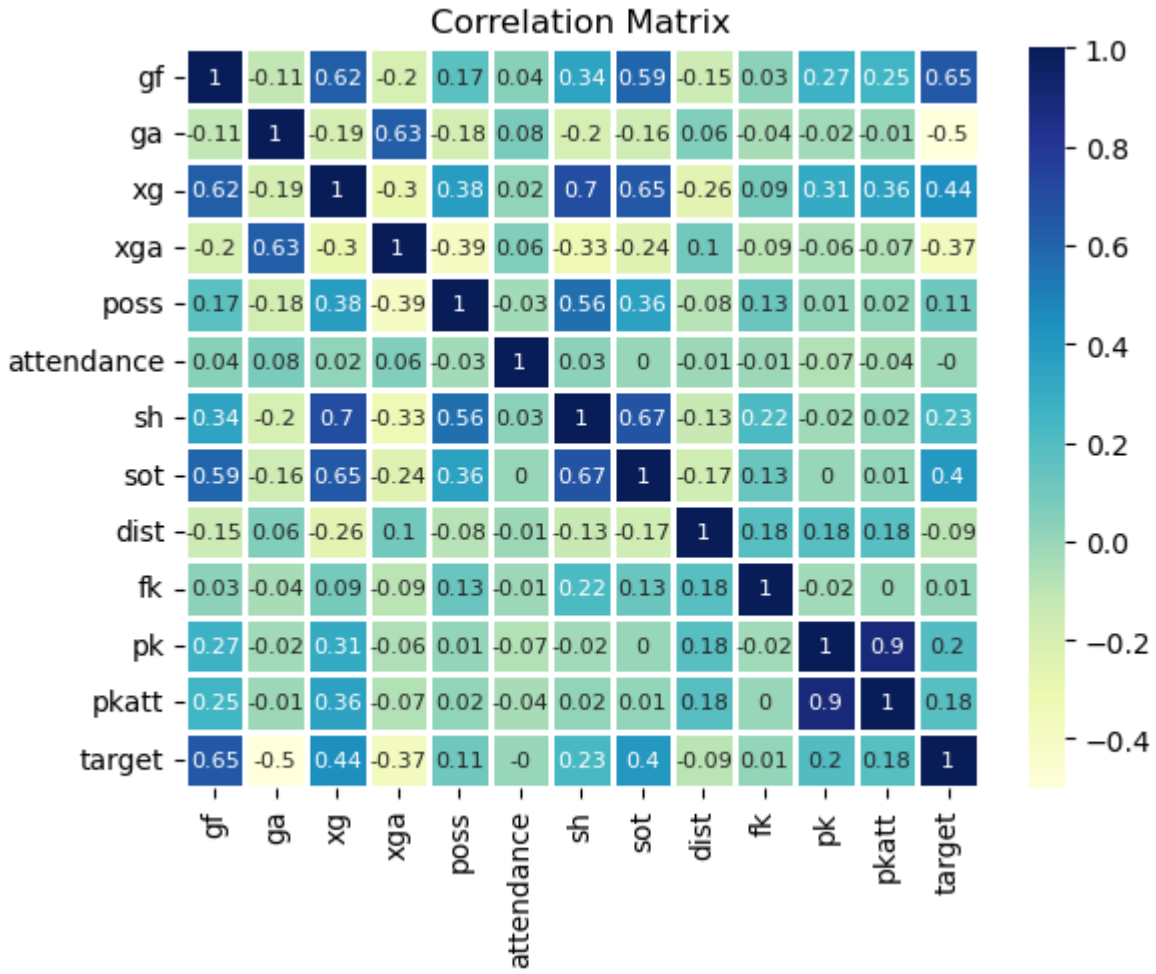


The analysis indicates that attendance is not a reliable predictor for the given project. However, it suggests that goals scored and goals conceded could be viable factors for prediction.

In [380…

```
import seaborn as sns

# Select the relevant columns from your football bundesliga dataset
selected_columns = ['gf', 'ga', 'xg', 'xga', 'poss', 'attendance', 'sh', 'sc
subset_data = bundesliga[selected_columns]

# Compute the correlation matrix
correlation_matrix = subset_data.corr().round(2)
```

```
# Plot the correlation matrix as a heatmap
sns.heatmap(correlation_matrix, cmap='YlGnBu', linewidths=1, annot=True, anr
plt.title('Correlation Matrix')
plt.show()
```



Based on the visualizations, we can conclude that venue code, opp code, hour, and day code are potential predictors that can be used to train the model effectively. These variables show significant patterns and variations in relation to the target variable, making them suitable features for predicting the outcome in the project.

In [381…
```
predictors = ["venue_code", "opp_code", "hour", "day_code"]
```

In [382…
```
rf.fit(train[predictors], train["target"])
```

Out[382]:
```
RandomForestClassifier
RandomForestClassifier(min_samples_split=10, n_estimators=50, rando
m_state=1)
```

In [383…
```
preds = rf.predict(test[predictors])
```

In [384…
```
# what percentage of time was prediction accurate?
```

```
from sklearn.metrics import accuracy_score
```

In [385…
```
error = accuracy_score(test["target"], preds)
```

In [386…
```
error

# The given code snippet is training a Random Forest Classifier model on the
```

Out[386]:
```
0.6268115942028986
```

In [387…
```
combined = pd.DataFrame(dict(actual=test["target"], predicted=preds))
```

In [388…
```
pd.crosstab(index=combined["actual"], columns=combined["predicted"])
```

Out[388]:

actual	predicted 0	1
0	139	33
1	70	34

In [389…
```
# Decent prediction, oculd be improved
33/(33+34)
```

Out[389]:
```
0.4925373134328358
```

3.2 Analysis using Accuracy and Precision

The accuracy of the model on the test set is 63%, indicating that it correctly predicted the match result (win/loss) in 63% of the cases. However, when specifically considering the precision score, which measures the proportion of correctly predicted positive results (wins), the model achieved a precision of around 49%. This suggests that while the model has an overall decent accuracy, its precision in predicting wins is relatively lower, with incorrect predictions made about 50% of the time. (When we predicted loss we were correct 139 times, wrong 70 times however when we predicted win we were wrong about 50% of the times, therefore this model could be improved alot and is not a good predictor currently.)

3.3 Revising the Model

We could design a function like `moving_averages` that calculates moving averages for a specified set of columns within a group.

1. The function first sorts the group by date in ascending order.
2. It then calculates the moving average for the specified columns using a window size of 3, with the option to close the left side of the window.
3. If the number of columns in the moving average matches the number of new columns specified, the function adds the moving average values to the group as

new columns, drops any rows with missing values in the new columns, and returns the modified group.

In [390…
```
# Revising our accuracy metric
# precision score

from sklearn.metrics import precision_score

precision_score(test["target"], preds)
```

Out[390]: 0.5074626865671642

In [391…
```
# splitting by team, metrics by team

grouped_bundesliga = bundesliga.groupby("team")
```

In [392…
```
group = grouped_bundesliga.get_group("Borussia Dortmund").sort_values("date"
```

In [393…
```
group
```

Out[393]:

	date	time	round	day	venue	result	gf	ga	opponent	xg	…	pk
629	2020-09-21	20:15	Matchweek 2	Mon	Away	W	3.0	1.0	Eintracht Frankfurt	1.9	…	1.0
630	2020-09-27	16:30	Matchweek 3	Sun	Home	L	2.0	5.0	VfB Stuttgart	0.9	…	0.0
631	2020-10-03	17:30	Matchweek 4	Sat	Away	D	1.0	1.0	VfL Wolfsburg	1.5	…	0.0
632	2020-10-17	17:30	Matchweek 5	Sat	Home	W	1.0	0.0	RB Leipzig	1.5	…	0.0
633	2020-10-24	12:30	Matchweek 6	Sat	Away	D	1.0	1.0	Borussia Mönchengladbach	1.1	…	0.0
…	…	…	…	…	…	…	…	…	…	…	…	…
28	2022-03-14	20:00	Matchweek 29	Mon	Away	D	0.0	0.0	TSG 1899 Hoffenheim	2.3	…	0.0
29	2022-04-02	15:00	Matchweek 31	Sat	Away	W	2.0	0.0	FC Union Berlin	1.8	…	0.0
30	2022-04-10	16:30	Matchweek 32	Sun	Home	D	2.0	2.0	VfL Bochum	2.0	…	0.0
31	2022-04-20	20:00	Matchweek 30	Wed	Home	W	3.0	0.0	SC Freiburg	1.2	…	0.0
32	2022-04-23	15:00	Matchweek 34	Sat	Home	W	5.0	1.0	SV Darmstadt 98	3.0	…	1.0

71 rows × 31 columns

In [394…

```
# latest 3 weeks use for the future week

#def moving_averages(group, cols, new_cols):
#    group = group.sort_values("date")
#    moving_stats = group[cols].rolling(3, closed='left').mean()
#    group[new_cols] = moving_stats
#    #group[new_cols] = moving_stats.values  # Assign the values instead of
#    group = group.dropna(subset=new_cols)
#    return group

def moving_averages(group, cols, new_cols):
    group = group.sort_values("date")
    moving_stats = group[cols].rolling(3, closed='left').mean()

    if len(moving_stats.columns) == len(new_cols):
        group[new_cols] = moving_stats.values
        group = group.dropna(subset=new_cols)

    return group
```

In [395…

```
cols = ["gf", "ga", "sh", "sot", "dist", "fk", "pk", "pkatt"]
new_cols = [f"{c}" for c in cols]

moving_averages(group, cols, new_cols)
```

Out[395]:

	date	time	round	day	venue	result	gf	ga	opponent
632	2020-10-17	17:30	Matchweek 5	Sat	Home	W	2.000000	2.333333	RB Leipzig
633	2020-10-24	12:30	Matchweek 6	Sat	Away	D	1.333333	2.000000	Borussia Mönchengladbach
634	2020-10-31	12:30	Matchweek 7	Sat	Away	W	1.000000	0.666667	SC Paderborn
635	2020-11-08	16:30	Matchweek 8	Sun	Home	D	1.000000	0.333333	VfL Bochum
636	2020-11-21	17:30	Matchweek 9	Sat	Away	L	1.000000	0.666667	Bayer 04 Leverkusen
...	...	...	...	...	...	...	...	...	...
28	2022-03-14	20:00	Matchweek 29	Mon	Away	D	2.333333	1.333333	TSG 1899 Hoffenheim
29	2022-04-02	15:00	Matchweek 31	Sat	Away	W	1.666667	0.333333	FC Union Berlin
30	2022-04-10	16:30	Matchweek 32	Sun	Home	D	2.000000	0.333333	VfL Bochum
31	2022-04-20	20:00	Matchweek 30	Wed	Home	W	1.333333	0.666667	SC Freiburg
32	2022-04-23	15:00	Matchweek 34	Sat	Home	W	2.333333	0.666667	SV Darmstadt 98

68 rows × 31 columns

In [396…
```
bundesliga_moving = bundesliga.groupby("team").apply(lambda x: moving_averag
```

In [397…
```
bundesliga_moving
```

Out[397]:

team		date	time	round	day	venue	result	gf	ga	oppo
Arminia Bielefeld	**1316**	2020-10-04	12:00	Matchweek 4	Sun	Away	L	1.666667	3.666667	FC
	1317	2020-10-19	17:30	Matchweek 5	Mon	Home	D	1.666667	3.333333	FC
	1318	2020-10-26	17:30	Matchweek 6	Mon	Away	D	1.000000	1.666667	Fre
	1319	2020-11-02	17:30	Matchweek 7	Mon	Away	L	0.333333	1.000000	Aug
	1320	2020-11-08	12:00	Matchweek 8	Sun	Home	L	0.333333	1.000000	Bay Leverl
...	**...**	...	...	...	...	...	...	...	...	
VfL Wolfsburg	**493**	2022-03-13	14:00	Matchweek 29	Sun	Home	W	0.000000	2.666667	Ho
	494	2022-03-18	20:00	Matchweek 30	Fri	Away	W	0.666667	1.666667	Eint Frar
	495	2022-04-02	15:00	Matchweek 31	Sat	Home	D	1.666667	2.000000	FC
	496	2022-04-09	15:00	Matchweek 32	Sat	Away	W	2.000000	1.333333	Darm
	497	2022-04-25	20:00	Matchweek 34	Mon	Away	D	2.333333	1.000000	TSG Hoffer

1317 rows × 31 columns

In [398…
```
bundesliga_moving = bundesliga_moving.droplevel('team')
```

In [399…
```
bundesliga_moving
```

Out[399]:

	date	time	round	day	venue	result	gf	ga	opponent	xg
1316	2020-10-04	12:00	Matchweek 4	Sun	Away	L	1.666667	3.666667	FC Koln	0.1
1317	2020-10-19	17:30	Matchweek 5	Mon	Home	D	1.666667	3.333333	FC Union Berlin	1.0
1318	2020-10-26	17:30	Matchweek 6	Mon	Away	D	1.000000	1.666667	SC Freiburg	0.6
1319	2020-11-02	17:30	Matchweek 7	Mon	Away	L	0.333333	1.000000	FC Augsburg	0.5
1320	2020-11-08	12:00	Matchweek 8	Sun	Home	L	0.333333	1.000000	Bayer 04 Leverkusen	0.8
...	...	...	...	...	...	...	...	...	...	...
493	2022-03-13	14:00	Matchweek 29	Sun	Home	W	0.000000	2.666667	Holstein Kiel	2.5
494	2022-03-18	20:00	Matchweek 30	Fri	Away	W	0.666667	1.666667	Eintracht Frankfurt	2.2
495	2022-04-02	15:00	Matchweek 31	Sat	Home	D	1.666667	2.000000	FC Koln	1.1
496	2022-04-09	15:00	Matchweek 32	Sat	Away	W	2.000000	1.333333	SV Darmstadt 98	1.0
497	2022-04-25	20:00	Matchweek 34	Mon	Away	D	2.333333	1.000000	TSG 1899 Hoffenheim	0.4

1317 rows × 31 columns

In [400…

```
bundesliga_moving.index = range(bundesliga_moving.shape[0])
```

3.4 Predicting the future using our machine learning

The opponent code variable in my project exhibits non-linear patterns, and utilizing a random forest algorithm will enable me to capture and identify these non-linearities effectively. By employing random forest, I can also uncover complex relationships and correlations between the opponent code and other variables in the dataset, leading to more accurate predictions and improved model performance.

In [401…

```
# Predicting the Future - using the model


def make_predictions(data, predictors):
    train = data[data["date"] < '2022-01-01']
    test = data[data["date"] > '2022-01-01']
    rf.fit(train[predictors], train["target"])
    preds = rf.predict(test[predictors])
    combined = pd.DataFrame(dict(actual=test["target"], predicted=preds), ir
```

```
    error = precision_score(test["target"], preds)
    return combined, error
```

In [402…]
```
combined, error = make_predictions(bundesliga_moving, predictors + new_cols)
```

In [403…]
```
# improved precision around 20%

error
```

Out[403]: 0.5

In [404…]
```
combined = combined.merge(bundesliga_moving[["date", "team", "opponent", "re
```

In [405…]
```
combined.head(10)
```

Out[405]:

	actual	predicted	date	team	opponent	result
88	1	1	2022-01-19	Bayer 04 Leverkusen	VfB Stuttgart	W
89	0	1	2022-01-23	Bayer 04 Leverkusen	FC Nurnberg	L
90	0	1	2022-02-09	Bayer 04 Leverkusen	FC Koln	L
91	0	0	2022-02-13	Bayer 04 Leverkusen	Eintracht Frankfurt	L
92	1	0	2022-02-19	Bayer 04 Leverkusen	Borussia Dortmund	W
93	0	0	2022-02-23	Bayer 04 Leverkusen	FC Union Berlin	L
94	1	0	2022-02-26	Bayer 04 Leverkusen	VfL Wolfsburg	W
95	1	1	2022-03-07	Bayer 04 Leverkusen	FSV Mainz 05	W
96	0	0	2022-03-12	Bayer 04 Leverkusen	FC Bayern Munich	L
97	1	0	2022-03-16	Bayer 04 Leverkusen	SC Freiburg	W

In [406…]
```
# make the names consisitent

class Replacers(dict):
    __missing__ = lambda self, key: key

map_values = {"SC Freiburg": "SC Freiburg", "FC Bayern Munich": "FC Bayern M
mapping = Replacers(**map_values)
```

In [407…]
```
combined["new_team"] = combined["team"].map(mapping)
```

In [408…]
```
merged = combined.merge(combined, left_on=["date", "new_team"], right_on=["o
```

Matching predictions on both sides.

In [409…]
```
merged
```

Out[409]:

	actual_x	predicted_x	date	team_x	opponent_x	result_x	new_team_x	actual_
0	1	1	2022-01-19	Bayer 04 Leverkusen	VfB Stuttgart	W	Bayer 04 Leverkusen	
1	0	1	2022-01-23	Bayer 04 Leverkusen	FC Nurnberg	L	Bayer 04 Leverkusen	
2	0	1	2022-02-09	Bayer 04 Leverkusen	FC Koln	L	Bayer 04 Leverkusen	
3	0	0	2022-02-13	Bayer 04 Leverkusen	Eintracht Frankfurt	L	Bayer 04 Leverkusen	
4	1	0	2022-02-19	Bayer 04 Leverkusen	Borussia Dortmund	W	Bayer 04 Leverkusen	
...	...	...	...	...	...	...	...	.
257	1	0	2022-03-13	VfL Wolfsburg	Holstein Kiel	W	VfL Wolfsburg	
258	1	0	2022-03-18	VfL Wolfsburg	Eintracht Frankfurt	W	VfL Wolfsburg	
259	0	0	2022-04-02	VfL Wolfsburg	FC Koln	D	VfL Wolfsburg	
260	1	0	2022-04-09	VfL Wolfsburg	SV Darmstadt 98	W	VfL Wolfsburg	
261	0	0	2022-04-25	VfL Wolfsburg	TSG 1899 Hoffenheim	D	VfL Wolfsburg	

262 rows × 13 columns

In [410…
```
merged[(merged["predicted_x"] == 1) & (merged["predicted_y"] ==0)]["actual_x
```

Out[410]:
```
1    20
0    18
Name: actual_x, dtype: int64
```

3.5 Re-Analysis using Accuracy and Precision

The accuracy of the model on the test set is now around 70%, indicating that it correctly predicted the match result (win/loss) in 70% of the cases.

In [411…
```
27/40
```

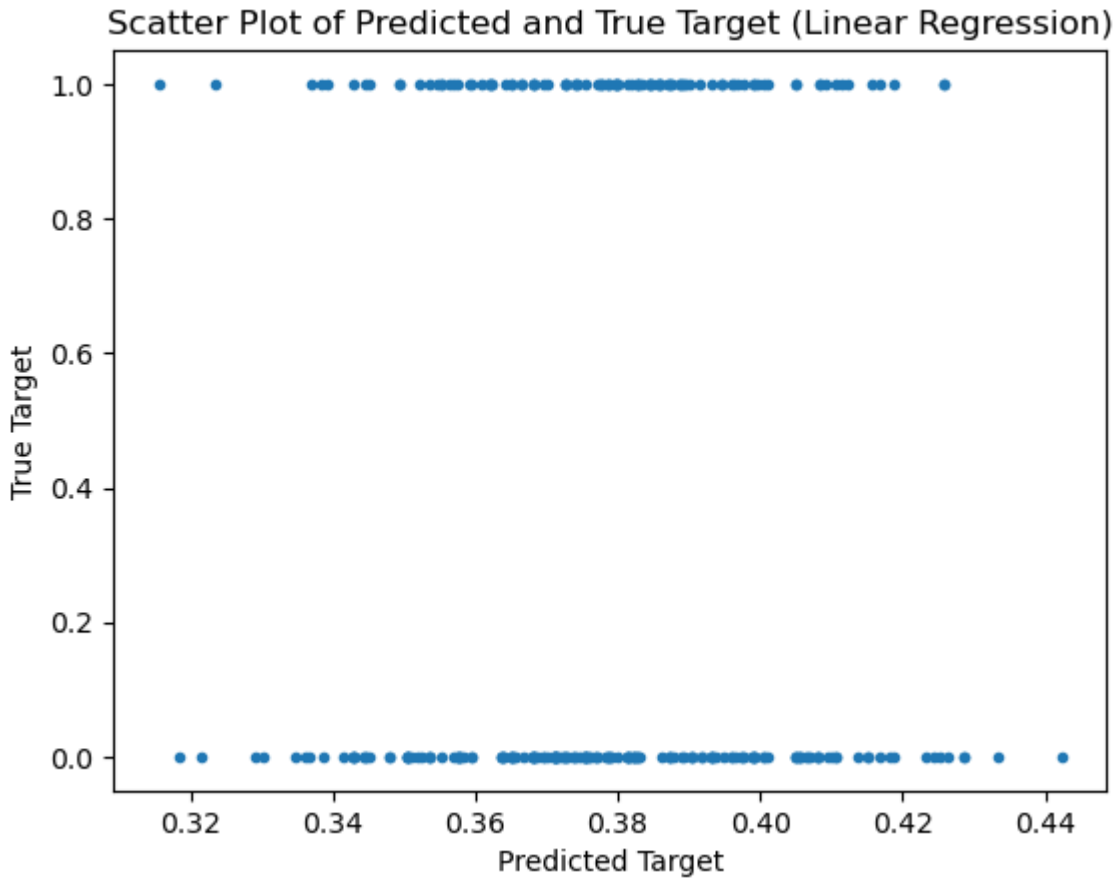
Out[411]:
```
0.675
```

3.7 Comparing few other ML Models

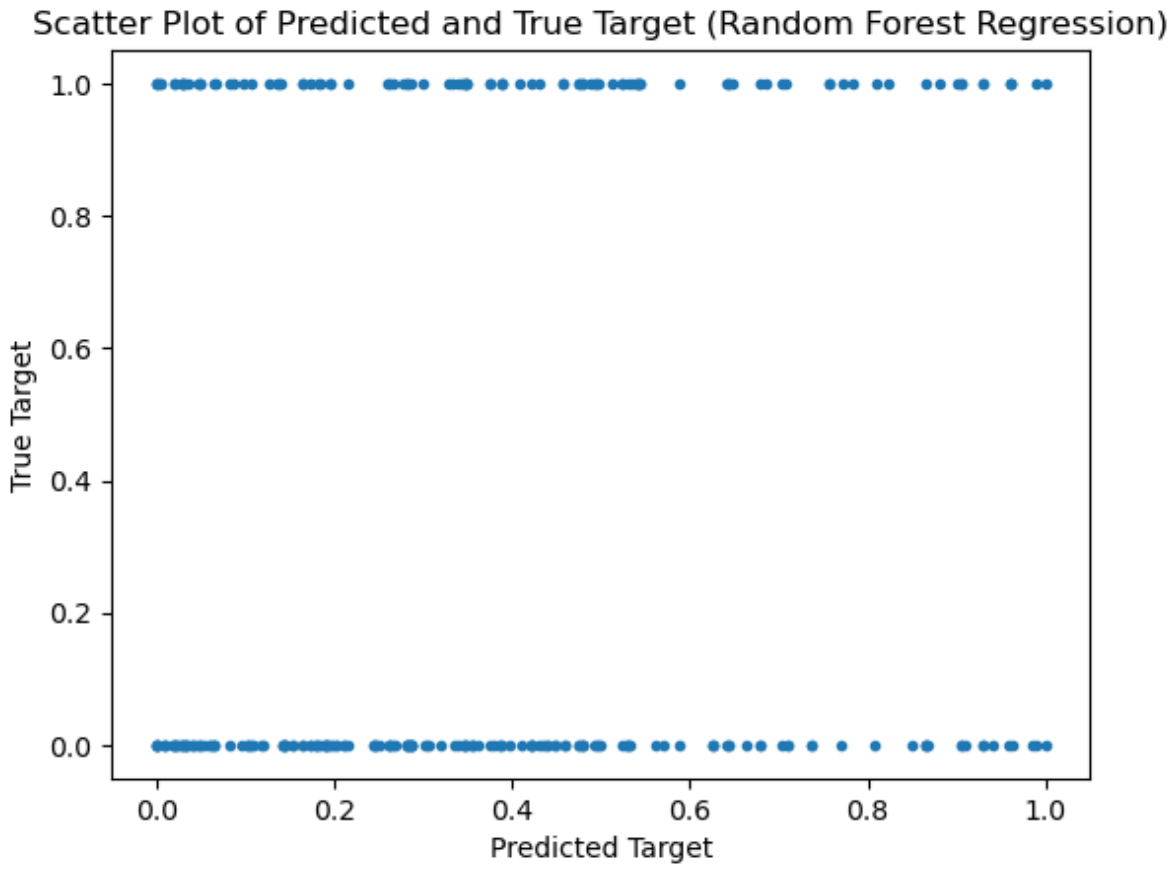
In [412…

```
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Linear Regression
lm = linear_model.LinearRegression()
x_train = train[['venue_code', 'opp_code', 'hour', 'day_code']]
y_train = train['target']
lm.fit(x_train, y_train)

x_test = test[['venue_code', 'opp_code', 'hour', 'day_code']]
y_test = test['target']
predict_target_lm = lm.predict(x_test)

rmse_lm = np.sqrt(mean_squared_error(predict_target_lm, y_test))
print("RMSE (Linear Regression):", rmse_lm)

# Scatter Plot
plt.plot(predict_target_lm, y_test, '.')
plt.xlabel("Predicted Target")
plt.ylabel("True Target")
plt.title("Scatter Plot of Predicted and True Target (Linear Regression)")
plt.show()

# Random Forest Regression
rf = RandomForestRegressor()
rf.fit(x_train, y_train)

predict_target_rf = rf.predict(x_test)

rmse_rf = np.sqrt(mean_squared_error(predict_target_rf, y_test))
print("RMSE (Random Forest Regression):", rmse_rf)

# Scatter Plot
plt.plot(predict_target_rf, y_test, '.')
plt.xlabel("Predicted Target")
plt.ylabel("True Target")
plt.title("Scatter Plot of Predicted and True Target (Random Forest Regressi
plt.show()
```

```
RMSE (Linear Regression): 0.4849664572950414
```

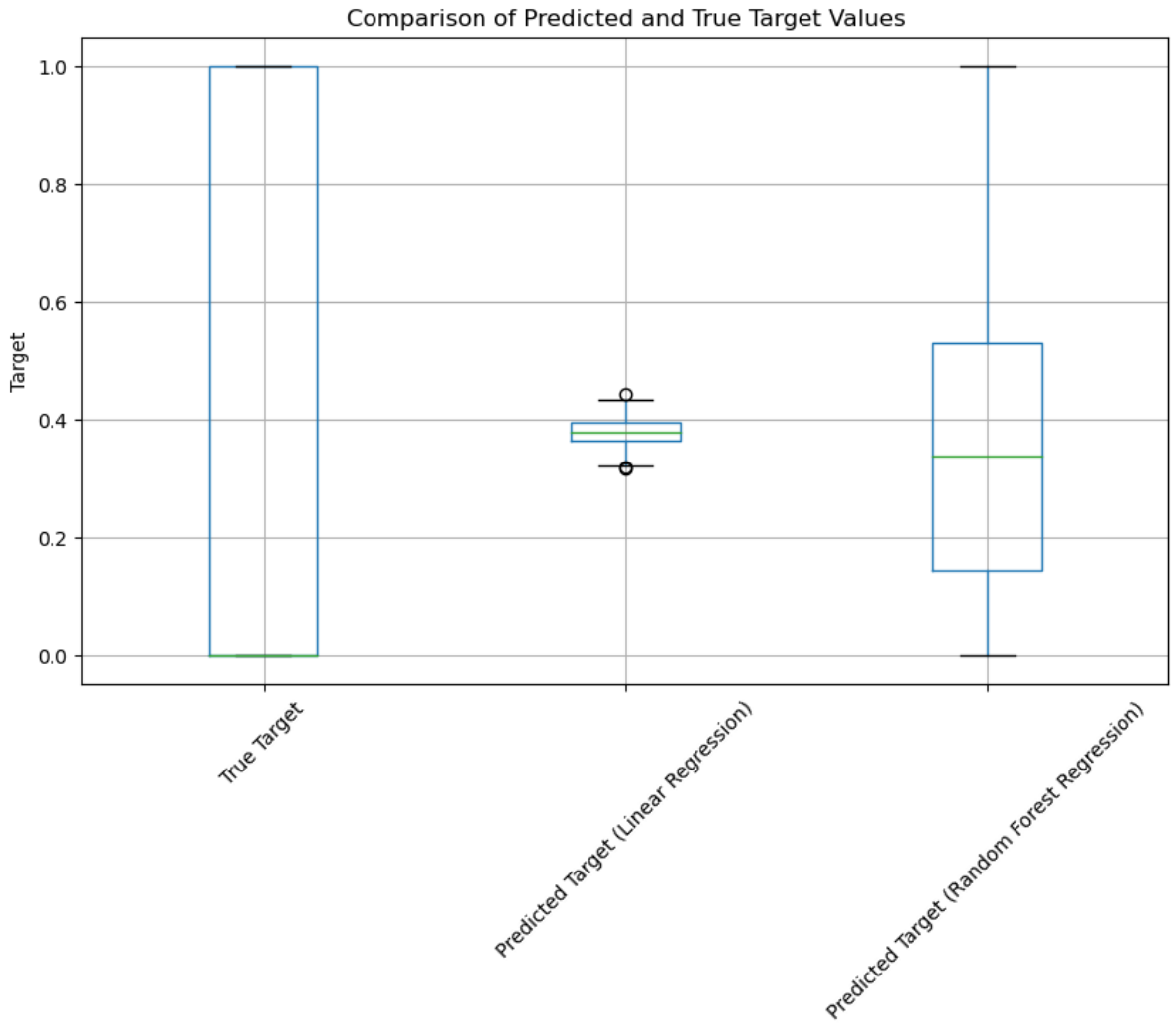



```
RMSE (Random Forest Regression): 0.530528319945578
```



In [413…

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Create a DataFrame for comparison
df_comparison = pd.DataFrame({'True Target': y_test, 'Predicted Target (Line

# Generate box plot
plt.figure(figsize=(10, 6))
df_comparison.boxplot()
plt.ylabel("Target")
plt.title("Comparison of Predicted and True Target Values")
plt.xticks(rotation=45)
plt.show()
```



In my project, I utilized the Random Forest model to predict the winner of the Bundesliga. This model outperformed the linear regression model because it has the ability to capture non-linear relationships within the football data. Unlike the linear regression model, which assumes a linear relationship between the predictors and the target variable, the Random Forest model can detect and incorporate complex interactions and non-linearities present in the data. As a result, it provides more

accurate predictions and a better understanding of the factors influencing the outcome of Bundesliga matches.