INTRODUCTION TO THE PROBLEM

In the world of soccer there are many different leagues, players, and teams. Winning games has more of an impact on teams than just being higher up in the score table. Winning games has a correlation with how much money a team will bring in so it is important for the board of a team to gauge and plan accordingly so they can operate within their financial means from season to season. The problem I would like to look into is how accurately can we predict a team to win any given match.

INTRODUCE THE DATA

The dataset that I am using is taken from five-thirty-eight's github repository 'data' specifically the 'soccer-spi'. Five-thirty-eight is an american website that focuses on data analysis revolving around a wide variety of topics including sports. This is a dataset that includes metrics from tens of thousands of professional soccer matches played globally. Below I included a list of each column that comes with the dataset I am using and what each column means/represents. These descriptions are taken from the github repo 'soccer-spi' created by five-thirty-eight.

season: The season during which the match was played

date: The date of the match (YYYY-MM-DD)

league_id: A unique identifier for the league this match was played in

league: The name of the league this match was played in

team1: The home team's name

team2: The away team's name

spi1: The home team's overall SPI rating before the match

spi2: The away team's overall SPI rating before the match

prob1: The probability of the home team winning the match

prob2: The probability of the away team winning the match

probtie: The probability of match ending in a draw (if applicable)

proj_score1: The number of goals we expected the home team to score

proj_score2: The number of goals we expected the away team to score

importance1: The importance of the match for the home team (0-100)

importance2: The importance of the match for the away team (0-100)

score1: The number of goals scored by the home team

score2: The number of goals scored by the away team

xg1: The number of expected goals created by the home team

xg2: The number of expected goals created by the away team

nsxg1: The number of non-shot expected goals created by the home team

nsxg2: The number of non-shot expected goals created by the away team

adj_score1: The number of goals scored by the home team, adjusted for game state

adj_score2: The number of goals scored by the home team, adjusted for game state

PRE-PROCESSING THE DATA

First I cleaned the data by dropping any rows from the dataset that had N/A for any of the columns that I would be using the create the model for the prediction analysis. If there was a value was listed as N/A for a feature used in the model that is something that would cause the model to return an error. Also the dataset is missing a critical column I need to for testing if test results match the training data and this column is which team won the match. Do create this column I checked the score1 and score2 columns and compared them and then returned 'team1', 'team2', or 'draw' for the new column to show who won the match more easily.

DATA UNDERSTANDING/VISUALIZATION

We can see that the confusion matrix generated shows that the are rarely any true positives for guessing that team1 would win the game and team1 actually winning the game. What is most common is for the model either to predict team2 or a draw. I tried to fix this by removing features I used as X but this made the problem worse.

MODELING

When creating the Model I used all the features that I think would make sense for the model. These features were data that was available before the match which is critcal because if I used data from after the match had been concluded that would ruin the point of the prediction problem. These features I used were each teams performance index, the probability that each time would win and the probability that there would be a draw, as well as the projected score for each team, and lastly the importance of the match from each team.

EVALUATION

The resulting confusion matrix shows that there is definitely some problem with the prediction model. The model over predicts that team 2 will win and almost picks team 1 zero percent of the time. There are a fair amount of draw predictions but I would expect the results to be more varied and even as opposed to this. It is almost like the model has found that the best chance of predicting a result is to default choose team 2 and will switch its choice to a draw if there is enough evidence that the teams are evenly matched.

STORYTELLING

We can see here with the results that predicting a soccer matches outcome is incredibly hard over the course of one game. It would be much easier to predict the placing of a team in a league over the course of a full season, but having a model predict the result of a singular match that could go either when teams are even close to evenly matched is quite difficult.

IMPACT SECTION

The conclusion I have to come to from my results is that it is hard to predict the outcome of a game, but we can also do much better than 'just guessing' the result of a game. Guessing would have a 1/3 chance of predicting the correct outcome while the prediciction model created has 1/2 chance of predicting the correct outcome. This shows that the results of the game are hard to predict and that more often than not the result is somewhat random. It would be beneficial if I could find a way to get the model to predict team 1 as well with some accuracy of course. Overall, I believe there is some work to be done with the model to create a more varied approach.

REFERENCES

https://github.com/fivethirtyeight/data/tree/master/soccer-spi

https://chat.openai.com/

https://scikit-learn.org/stable/supervised_learning.html

https://stackoverflow.com/questions/40615021/cannot-import-sklearn-model-selection-in-scikit-learn

https://seaborn.pydata.org/

CODE

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
df = pd.read_csv("spi_matches.csv")
```

In [3]:
```
df_cleaned = df.dropna(subset=['spi1', 'spi2', 'prob1', 'prob2', 'probtie', 'proj_scor
```

In [4]:
```
def determine_winner(team_a_score, team_b_score):
    if team_a_score > team_b_score:
        return "team1"
    elif team_b_score > team_a_score:
        return "team2"
    else:
        return "draw"

df["Winner"] = df.apply(lambda row: determine_winner(row['score1'], row['score2']), ax

print(df["Winner"])
```

```
0        team1
1        team1
2         draw
3         draw
4        team2
         ...
68908     draw
68909     draw
68910     draw
68911     draw
68912     draw
Name: Winner, Length: 68913, dtype: object
```

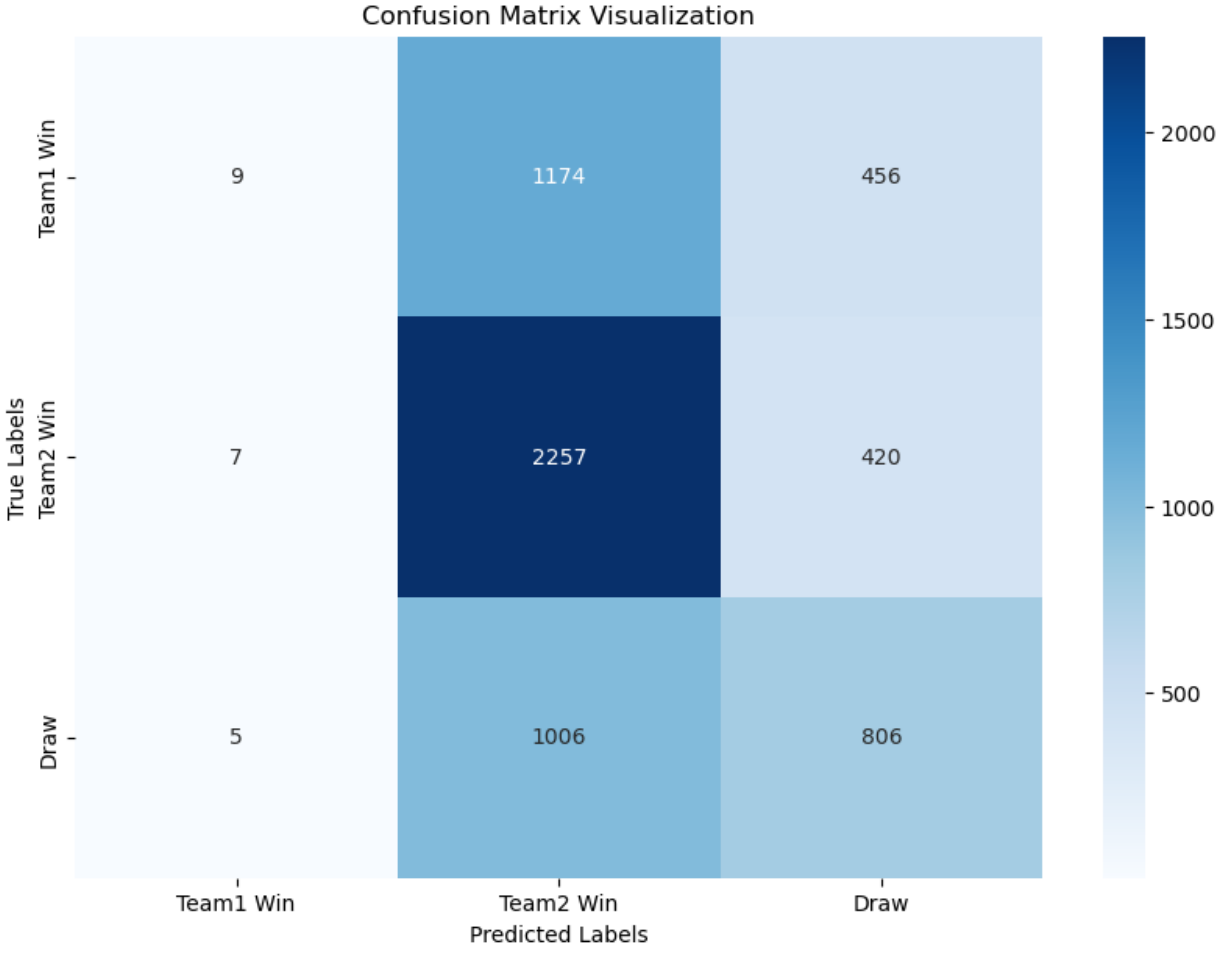
In [5]:
```
# primary features for the model itself
features = df[['spi1', 'spi2', 'prob1', 'prob2', 'probtie', 'proj_score1', 'proj_score

conditions = [
    (df['score1'] > df['score2']),
    (df['score1'] < df['score2'])
]
choices = [1,2]
df['outcome'] = np.select(conditions, choices, default=0)



X = features
y = df['outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_stat
X_train_cleaned = X_train.dropna()
y_train_cleaned = y_train.loc[X_train_cleaned.index]
X_test_cleaned = X_test.dropna()
y_test_cleaned = y_test.loc[X_test_cleaned.index]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_cleaned)
X_test_scaled = scaler.transform(X_test_cleaned)
```

In [6]:
```
model = LogisticRegression(random_state = 42)
model.fit(X_train_scaled, y_train_cleaned)
y_pred = model.predict(X_test_scaled)
```

In [7]:
```
accuracy = accuracy_score(y_test_cleaned, y_pred)
print(f"Accuracy: {accuracy:.4f}")

conf_matrix = confusion_matrix(y_test_cleaned, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")
```

```
Accuracy: 0.5003
Confusion Matrix:
[[   9 1174  456]
 [   7 2257  420]
 [   5 1006  806]]
```

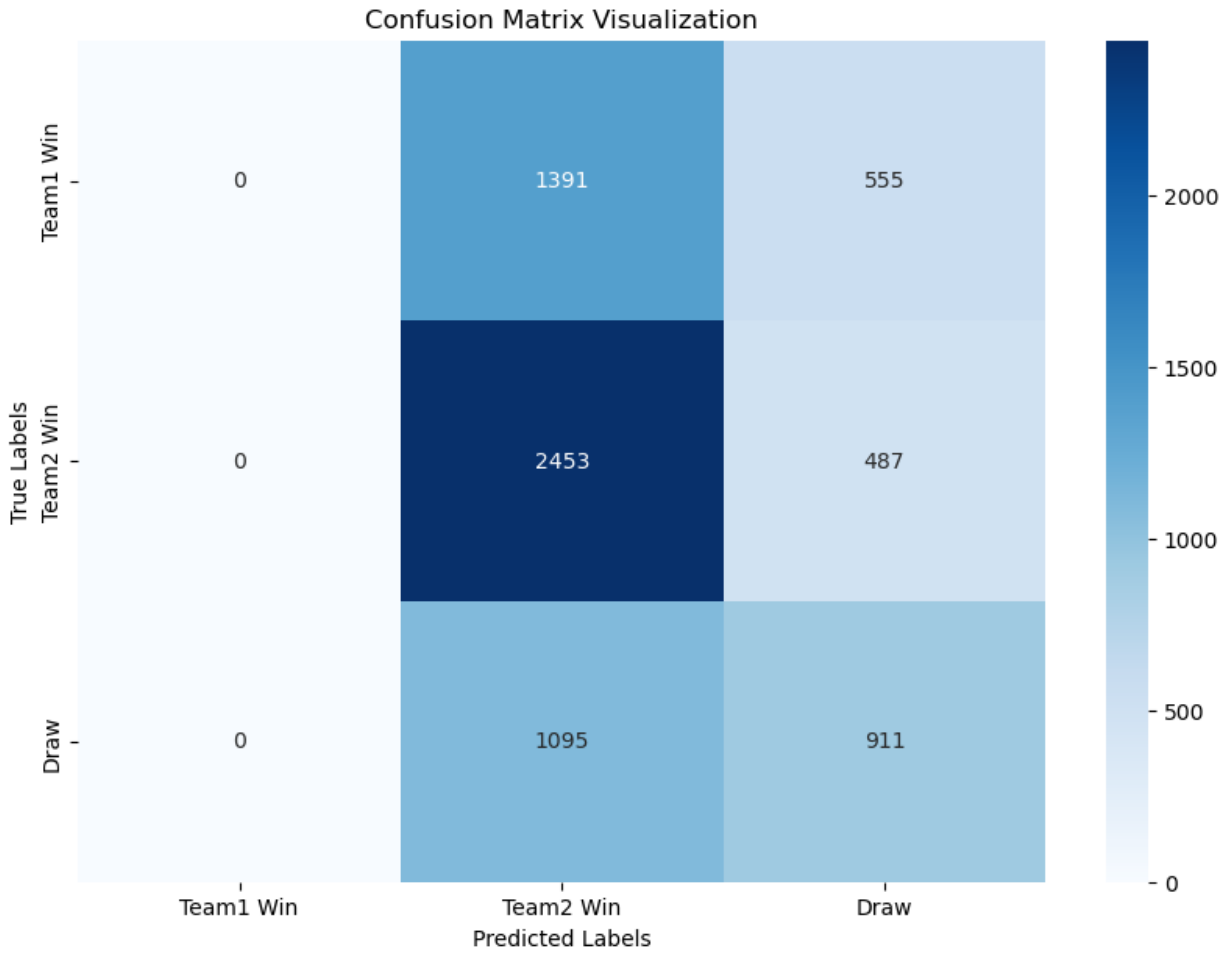
In [8]:
```
conf_matrix = confusion_matrix(y_test_cleaned, y_pred)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Team1 Win',

plt.title('Confusion Matrix Visualization')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [9]:
```
# primary features for the model itself
features = df[['spi1', 'spi2']]
```

```
conditions = [
    (df['score1'] > df['score2']),
    (df['score1'] < df['score2'])
]
choices = [1,2]
df['outcome'] = np.select(conditions, choices, default=0)



X = features
y = df['outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_stat
X_train_cleaned = X_train.dropna()
y_train_cleaned = y_train.loc[X_train_cleaned.index]
X_test_cleaned = X_test.dropna()
y_test_cleaned = y_test.loc[X_test_cleaned.index]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_cleaned)
X_test_scaled = scaler.transform(X_test_cleaned)

model = LogisticRegression(random_state = 42)
model.fit(X_train_scaled, y_train_cleaned)
y_pred = model.predict(X_test_scaled)

accuracy = accuracy_score(y_test_cleaned, y_pred)
print(f"Accuracy: {accuracy:.4f}")

conf_matrix = confusion_matrix(y_test_cleaned, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")

conf_matrix = confusion_matrix(y_test_cleaned, y_pred)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Team1 Win',

plt.title('Confusion Matrix Visualization')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
Accuracy: 0.4881
Confusion Matrix:
[[   0 1391  555]
 [   0 2453  487]
 [   0 1095  911]]
```



In [10]:
```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(random_state = 42)
model_rf.fit(X_train_scaled, y_train_cleaned)
y_pred_rf = model_rf.predict(X_test_scaled)

accuracy_rf = accuracy_score(y_test_cleaned, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf:.4f}")
```

```
Random Forest Accuracy: 0.4267
```

In [11]:
```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train_cleaned, y_train_cleaned)

# tree.plot_tree(clf)
```