Premier League Match & Table Predictor

Aim of the Project

🔗 **App link**: Streamlit App

This project aims to:

1. Predict the number of **goals scored** by each team in a Premier League match.
2. Use the predicted scores to assign **match points**.
3. Aggregate all matches to generate the **final Premier League table** (20 teams ranked by points).

Data Collection & Cleaning

Data was scraped from **fbref.com** using `pandas.read_html()`:

```
game_data = pd.read_html(
    'https://fbref.com/en/comps/9/schedule/Premier-League-Scores-and-Fixtures',
    attrs={'id': 'sched_2024-2025_9_1'}
)[0]
```

Cleaned using:

```
game_data.columns = game_data.columns.str.lower().str.replace(' ', '', regex=False)
game_data = game_data.dropna(how='all')
game_data['wk'] = game_data['wk'].astype(int)


game_data = game_data.rename(columns={
    'xg': 'xg_home',
    'xg.1': 'xg_away'
})

```

```

game_data = game_data.drop(columns=['venue','matchreport','notes'])

game_data
```

Final dataset shape: **(380, 11)** → matches expected for a full PL season.

Feature Engineering

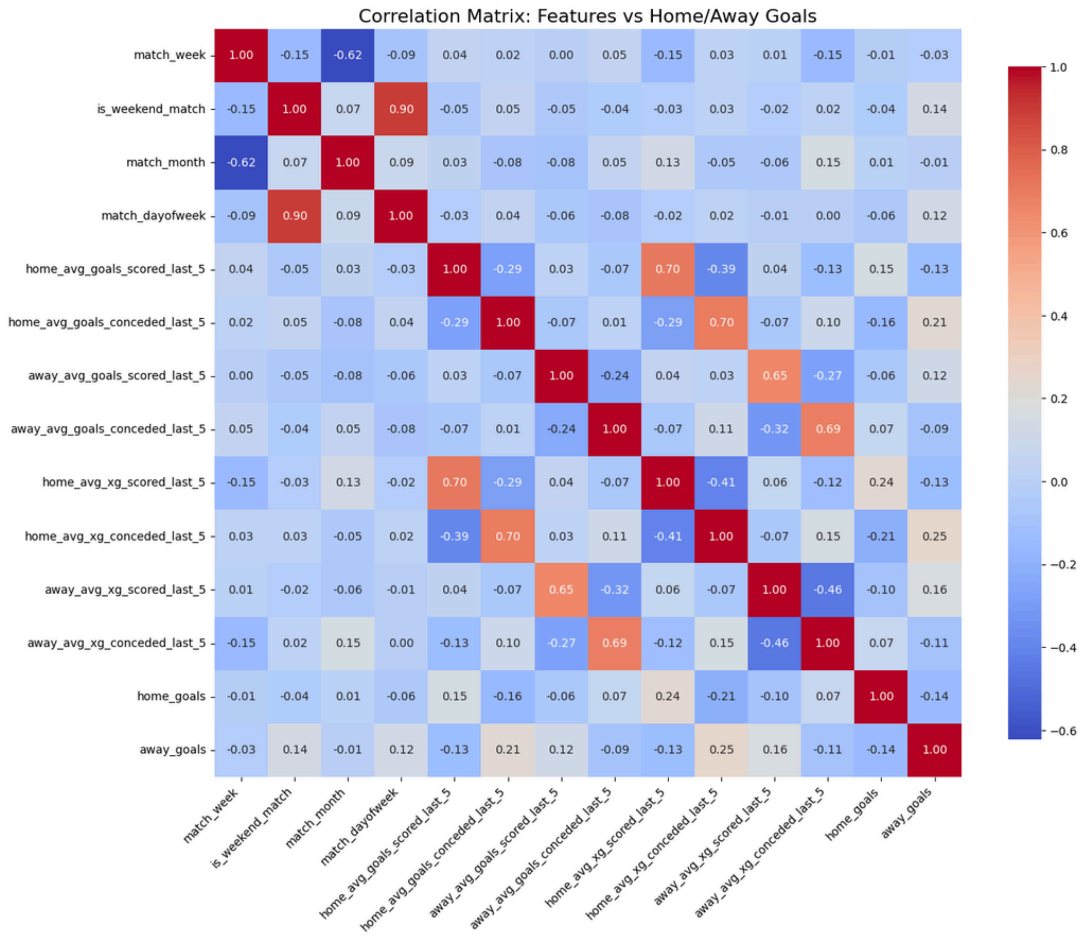
To capture team form, we created rolling averages of recent performance:

```python
def compute_team_rolling_stats(df, team_col, goals_for_col, goals_against_col, prefix):
    team_stats = []

    for team in df[team_col].unique():
        team_matches = df[(df[team_col] == team)].copy()
        team_matches = team_matches.sort_values('date')

        team_matches[f'{prefix}_avg_goals_scored_last_5'] = team_matches[goals_for_col].shift().rolling(5, min_periods=1).mean()
        team_matches[f'{prefix}_avg_goals_conceded_last_5'] = team_matches[goals_against_col].shift().rolling(5, min_periods=1).mean()

        team_stats.append(team_matches[[f'{prefix}_avg_goals_scored_last_5', f'{prefix}_avg_goals_conceded_last_5']])

    return pd.concat(team_stats).sort_index()
```

simirlar function was used for xG

Correlation analysis

A heatmap of correlations between engineered features and target variables (`home_goals`, `away_goals`) revealed:



Most Predictive Features (Good Correlation with Goals)

Correlated with `home_goals`:

Feature	Correlation
`home_avg_goals_scored_last_5`	**0.24**
`home_avg_xg_scored_last_5`	**0.23**
`home_avg_xg_conceded_last_5`	-0.21 (inverse relation)

Correlated with `away_goals`:

Feature	Correlation
`home_avg_xg_conceded_last_5`	**0.25**
`away_avg_xg_scored_last_5`	**0.16**
`away_avg_goals_scored_last_5`	**0.12**

Less Important Features (Low or No Correlation)

Feature	Correlation with `home_goals`	Correlation with `away_goals`

`match_week`	-0.01	-0.03
`match_month`	0.01	0.14
`is_weekend_match`	0.04	0.12
`match_dayofweek`	0.06	0.12

These were retained but weighted as weaker predictors.

Model Training (TensorFlow)

Input Preparation

- Categorical features (`home`, `away`) were one-hot encoded.
- Numerical features + encoded teams used as model input (`X_train`).
- Targets: `home_goals`, `away_goals`.

```
    home_enc = pd.get_dummies(features_train['home'], prefix='home')
    away_enc = pd.get_dummies(features_train['away'], prefix='away')
    X_train = pd.concat([features_train[feature_cols].fillna(0), home_enc, away_enc], axis=1)
    y_train = features_train[['home_goals', 'away_goals']].astype(float)
```

- **Input (X_train)**: Combines your numerical features with one-hot encoded home and away teams.
- **Target (y_train)**: The real number of home and away goals (used as regression targets).

Model architecture

```
        model = models.Sequential([
            layers.Dense(90, activation='relu', input_shape=(X_train.shape[1],)),
            layers.BatchNormalization(),
            layers.Dropout(0.2),
            layers.Dense(48, activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.2),
            layers.Dense(30, activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.2),
            layers.Dense(2)
        ])
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=int(verbose))
```

- 3 hidden layers: **90** → **48** → **30** units
- Each layer uses:
  - `ReLU` activation
  - `BatchNormalization` (improves stability)
  - `Dropout` (prevents overfitting)
- Final output layer has **2 nodes** to predict home and away goals

Model Compilation & Training

```
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mse', metrics=['mae'])
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=int(verbose))
```

- **Loss function**: Mean Squared Error (MSE) - standard for regression
- **Metric**: Mean Absolute Error (MAE) - helps track training progress
- Trained for **120 epochs** with **batch size 128**
- **20% of data** used for validation during training

Multiple Runs for Stability

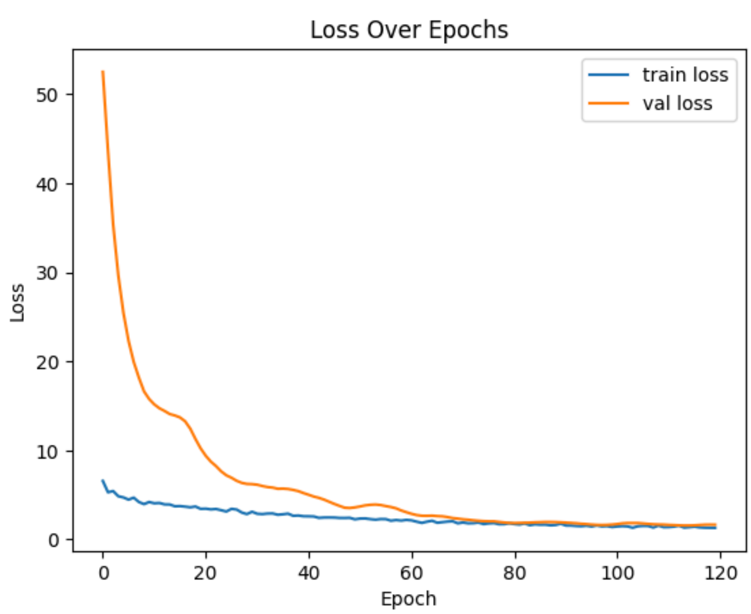
The model was trained **multiple times** (`n_runs = 10`, `50`, `100`, `500`) and results were averaged to:

- Reduce randomness from model initialization and dropout
- Provide a **stable and fair estimate** of match outcomes and final table

Model Accuracy

To ensure robustness, the model was trained **multiple times** and the predictions were **averaged**. This helps reduce the effect of randomness from model initialization and dropout layers.

The plot of **training vs validation loss** over 120 epochs gives a clear picture of how well the model learned:



Early Epochs (0–20):

- **Validation loss drops rapidly** → The model is quickly learning meaningful patterns.
- **Training loss decreases steadily** → Learning is effective.

Middle Epochs (20–60):

- Both losses continue to **decline smoothly**.
- No sign of **overfitting** - validation performance keeps improving.

Later Epochs (60–120):

- Loss curves **flatten out**, indicating convergence.
- **No sharp divergence** between training and validation → Model generalizes well.

Summary of Learning Behavior

- **Good convergence**: Model effectively learns from training data.
- **No overfitting**: Validation loss remains low and stable.
- **Final MSE is very low**, suggesting that the model is accurately predicting goal outcomes on average.

Prediction Performance

The final predictions were evaluated against actual match outcomes:

Metric	Accuracy
Match Outcome (W/D/L)	56.25%
Exact Home Goals Predicted	36.96%
Exact Away Goals Predicted	43.24%

Key Observations

- The model does **well at predicting match results** (win/draw/loss), with an outcome accuracy over 56%.
- **Away goals** were predicted **more accurately** than home goals.
- Predicting the **exact number of goals** is inherently difficult due to football's unpredictability - but the model still provides **reliable estimations**.

Example Findings

Example Predictions

Match	Predicted Score	Predicted Outcome	Actual Score	Actual Outcome	Result Type
Arsenal (Home) vs Tottenham (Away)	2 - 1 *(2.09 - 1.03)*	Home Win	2 - 1	Home Win	Accurate
Arsenal (Home) vs Man City (Away)	2 - 1 *(2.25 - 1.18)*	Home Win	5 - 1	Home Win	Partial Miss

- The model often **predicts the outcome correctly**, even when the **exact scoreline varies**.
- This consistency in **outcome prediction** makes it well-suited for **league table forecasting**.