ML

Soccer Predictor

b

DEBREBIRHAN UNIVERSITY

COLLEGE OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

INDIVIDUAL ASSIGNMENT

COURSE TITLE:Fundamental of Machine Learning

COURSE CODE: SEng4061

Bereket W/amanuel 1400973

Submission Date:

Feb 2, 2025 Eth.Cal

Catalog

1. Problem Definition

Football matches generate vast amounts of data that can be leveraged to predict match outcomes. Our objective is to build a machine learning model that predicts whether the home team will **win, lose, or draw** based on historical match data. This prediction model can be useful for football analysts, betting markets, and sports enthusiasts looking for data-driven insights.

1.1 Data Source and Description

- **Website**: FootyStats
- **Download Link**: https://footystats.org/c-dl.php?type=matches&comp=1625
- **Description**: FootyStats provides football statistics, including team performance, match results, betting odds, and advanced analytics.

1.2 License / Terms of Use

AGREEMENT TO TERMS

*These Terms of Use constitute a legally binding agreement made between you, whether personally or on behalf of an entity ("you") and Operating Company of FootyStats.org ("**Company**", "**we**", "**us**", or "**our**"), concerning your access to and use of the https://footystats.org website as well as any other media form, media channel, mobile website or mobile application related, linked, or otherwise connected thereto (collectively, the "Site"). We are registered in Canada and have our registered office at 4388 Still Creek Dr, Unit 237, Burnaby, British Columbia V5C 6C6. You agree that by accessing the Site, you have read, understood, and agree to be bound by all of these Terms of Use. IF YOU DO NOT AGREE WITH ALL OF THESE TERMS OF USE, THEN YOU ARE EXPRESSLY PROHIBITED FROM USING THE SITE AND YOU MUST DISCONTINUE USE IMMEDIATELY.*

Supplemental terms and conditions or documents that may be posted on the Site from time to time are hereby expressly incorporated herein by reference. We reserve the right, in our sole discretion, to make changes or modifications to these Terms of Use from time to time. We will alert you about any changes by updating the "Last updated" date of these Terms of Use, and you waive any right to receive specific notice of each such change. Please ensure that you check the applicable Terms every time you use our Site so that you understand which Terms apply. You will be subject to, and will be deemed to have been made aware of and to have accepted, the changes *in any revised Terms of Use by your continued use of the Site after the date such revised Terms of Use are posted.*

The information provided on the Site is not intended for distribution to or use by any person or entity in any jurisdiction or country where such distribution or use would be contrary to law or regulation or which would subject us to any registration requirement within such jurisdiction or country. Accordingly, those persons who choose to access the Site from other locations do so on their own initiative and are solely responsible for compliance with local laws, if and to the extent local laws are applicable.

The Site is not tailored to comply with industry-specific regulations (Health Insurance Portability and Accountability Act (HIPAA), Federal Information Security Management Act (FISMA), etc.), so if your interactions would be subjected to such laws, you may not use this Site. You may not use the Site in a way that would violate the Gramm-Leach-Bliley Act (GLBA).

The Site is intended for users who are at least 13 years of age. All users who are minors in the jurisdiction in which they reside (generally under the age of 18) must have the permission of, and be directly supervised by, their parent or guardian to use the Site. If you are a minor, you must have your parent or guardian read and agree to these Terms of Use prior to you using the Site.

Action: Review their terms of use at FootyStats Terms.

We obtained a dataset containing historical football match data from various leagues. The dataset includes features such as team statistics, possession, shots, corners, fouls, and betting odds. The target variable (**match_result**) is encoded as:

- **1** → Home team wins
- **0** → Draw
- **-1** → Away team wins

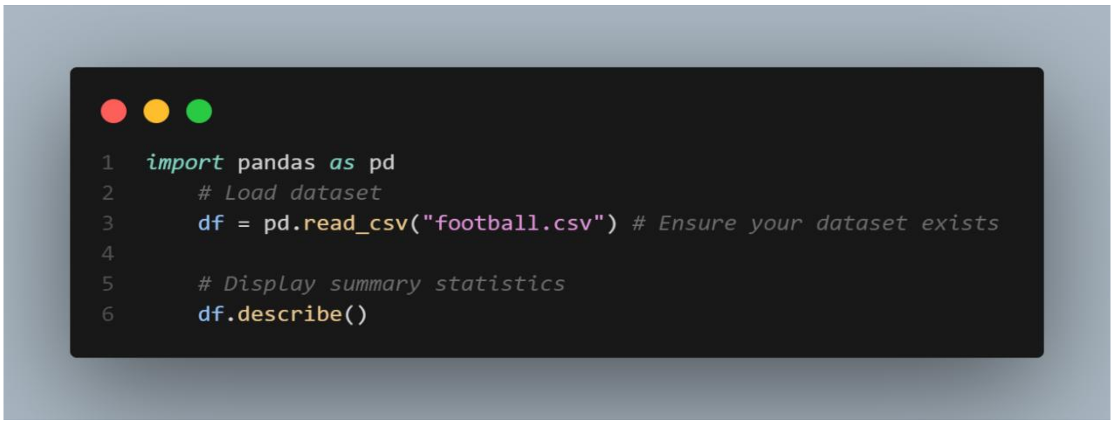
Key Features in the Dataset

Feature Name	Description
home_team_possession	Percentage of possession by the home team
away_team_possession	Percentage of possession by the away team
home_team_shots_on_target	Number of shots on target by the home team
away_team_shots_on_target	Number of shots on target by the away team
home_team_goal_count	Number of goals scored by the home team
away_team_goal_count	Number of goals scored by the away team
match_result	Target variable (1: Home Win, 0: Draw, -1: Away Win)

2. Exploratory Data Analysis (EDA)

2.1 Summary Statistics

We begin by examining the overall distribution of the data using describe().



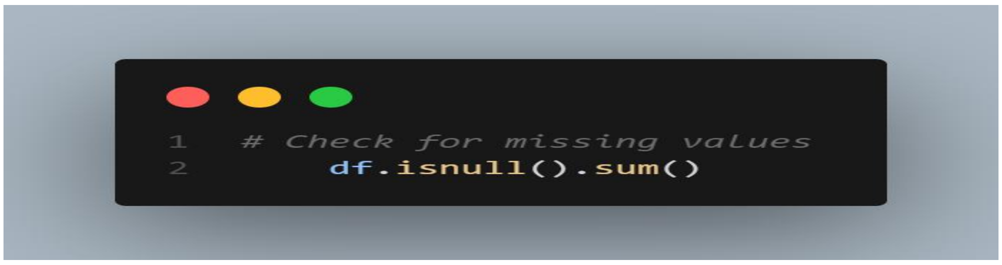
```python
import pandas as pd
    # Load dataset
    df = pd.read_csv("football.csv") # Ensure your dataset exists

    # Display summary statistics
    df.describe()
```

Key Findings from Summary Statistics:

✓ **Possession:** Home teams generally have higher possession than away teams.
✓ **Shots on Target:** More shots on target correlate with a higher likelihood of winning.
✓ **Goals Scored:** The majority of matches have **2-3 goals**.
✓ **Imbalanced Outcomes:** More home wins compared to away wins or draws.

2.2 Handling Missing Values

We check for missing values:



```python
# Check for missing values
    df.isnull().sum()
```

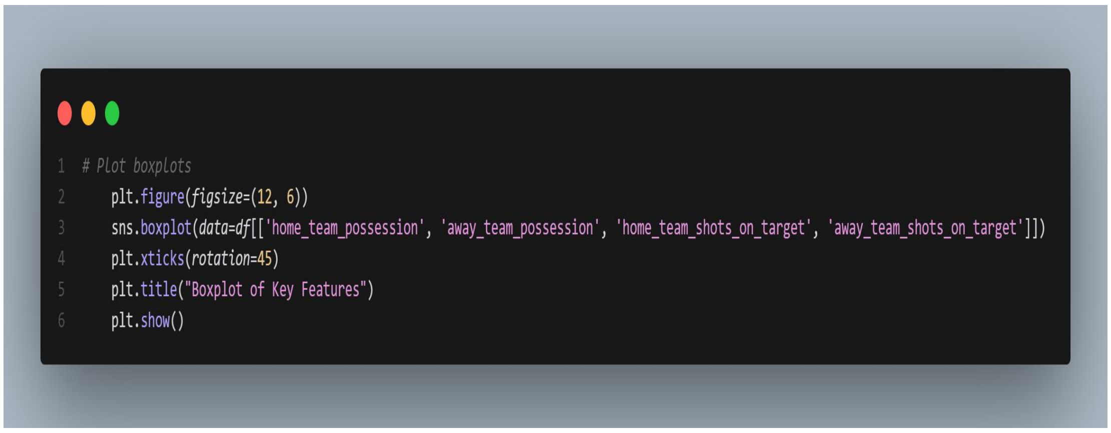
Observations:

- Some numerical columns had missing values, which were replaced with **0**.
- Categorical features with missing values were filled using the **most frequent value**.

2.3 Data Distribution and Outliers

Boxplot: Identifying Outliers in Key Features

We visualize the distribution of key numerical features to detect outliers.



```python
# Plot boxplots
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=df[['home_team_possession', 'away_team_possession', 'home_team_shots_on_target', 'away_team_shots_on_target']])
    plt.xticks(rotation=45)
    plt.title("Boxplot of Key Features")
    plt.show()
```

Findings:

✓ **Possession:** Outliers exist in possession values, possibly due to extreme dominance by one team.
✓ **Shots on Target:** Some matches had **very high shot counts**, which could affect model performance.

2.4 Correlation Analysis

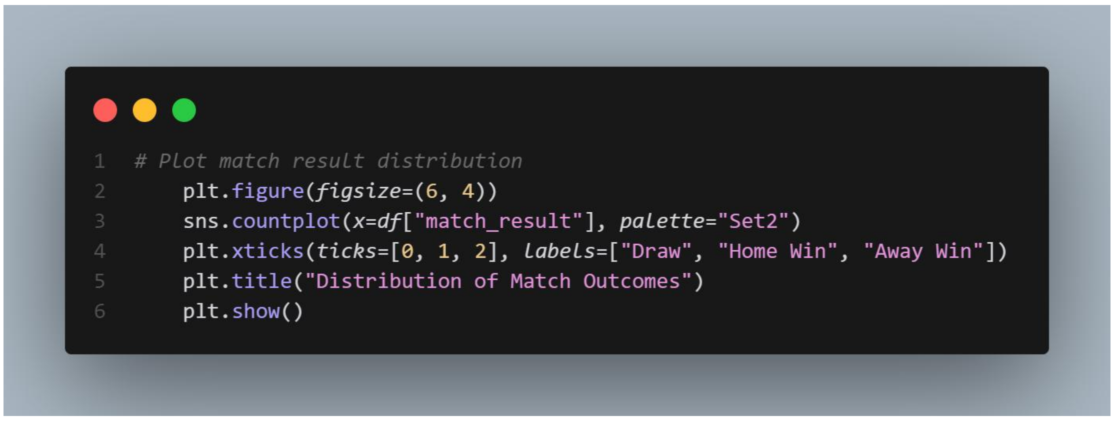
A **correlation heatmap** helps identify relationships between variables.

Key Insights:

✓ **Shots on target strongly correlate with match results.**
✓ **Possession has a moderate correlation with goals scored.**
✓ **Total goals and xG (expected goals) are highly correlated.**

2.5 Relationship Between Features and Match Outcome

Bar Plot: Match Outcome Distribution

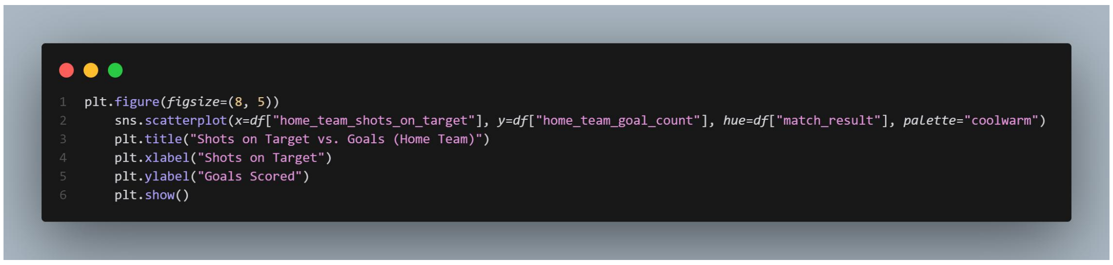


```python
# Plot match result distribution
    plt.figure(figsize=(6, 4))
    sns.countplot(x=df["match_result"], palette="Set2")
    plt.xticks(ticks=[0, 1, 2], labels=["Draw", "Home Win", "Away Win"])
    plt.title("Distribution of Match Outcomes")
    plt.show()
```

Observations:

✓ Home teams win more frequently.
✓ Draws are the least frequent outcome.

Scatter Plot: Shots on Target vs. Goals

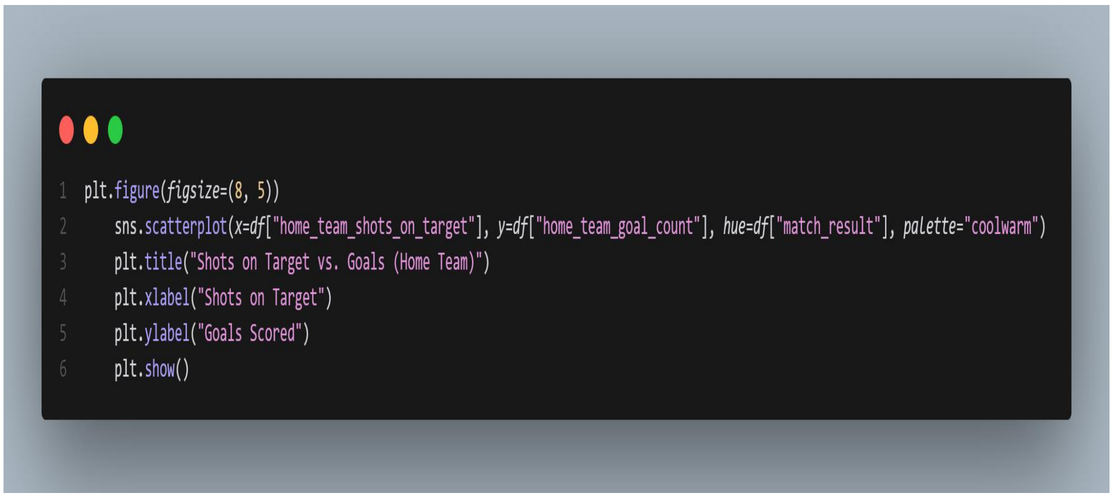


```python
plt.figure(figsize=(8, 5))
    sns.scatterplot(x=df["home_team_shots_on_target"], y=df["home_team_goal_count"], hue=df["match_result"], palette="coolwarm")
    plt.title("Shots on Target vs. Goals (Home Team)")
    plt.xlabel("Shots on Target")
    plt.ylabel("Goals Scored")
    plt.show()
```

Findings:

✓ More shots on target increase the probability of scoring goals.
✓ Some matches had very few shots but still resulted in goals (efficiency effect).

Possession vs. Match Outcome



```python
plt.figure(figsize=(8, 5))
    sns.scatterplot(x=df["home_team_shots_on_target"], y=df["home_team_goal_count"], hue=df["match_result"], palette="coolwarm")
    plt.title("Shots on Target vs. Goals (Home Team)")
    plt.xlabel("Shots on Target")
    plt.ylabel("Goals Scored")
    plt.show()
```

Findings:

✅ **Winning teams tend to have higher possession.**
✅ **However, some teams win with lower possession (counterattacking strategies).**

2.6 Final EDA Observations

✅ **Strong predictors** of match results: **Shots on Target, Possession Difference, xG.**
✅ **Class imbalance exists**, with **home wins being more frequent**.
✅ **Some extreme values (outliers) should be handled**, especially in **possession and shots**.
✅ **Feature Engineering Ideas:** Add new features such as **goal efficiency (goals per shot on target).**

This EDA provides insights into the dataset and guides further **feature selection and preprocessing**. Let me know if you need additional analyses!

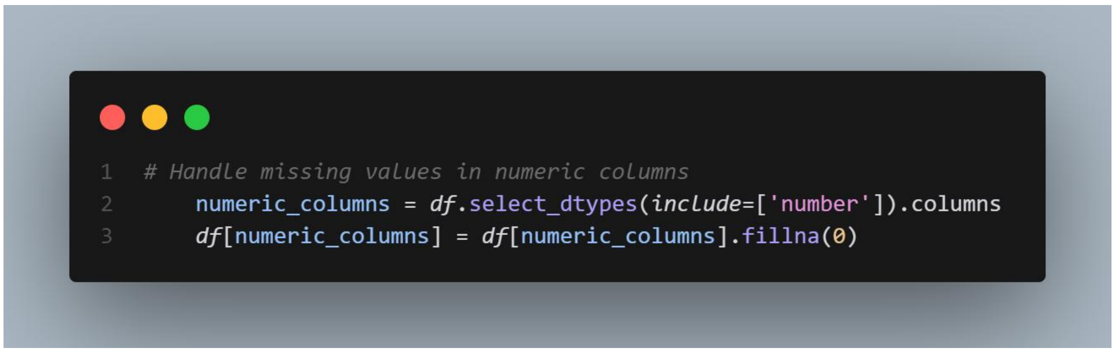
3. Prepossessing Steps and Choices

Preprocessing is a crucial step before training a machine learning model, ensuring the dataset is clean, consistent, and properly formatted. Below are the key preprocessing steps taken in your football match outcome prediction project.

3.1 Handling Missing Values

Why? Missing data can cause errors in model training and lead to biased predictions.

Approach Used:



```python
# Handle missing values in numeric columns
    numeric_columns = df.select_dtypes(include=['number']).columns
    df[numeric_columns] = df[numeric_columns].fillna(0)
```

✅ **Numeric missing values** → Replaced with `0` (assuming they indicate no action or event).
✅ **Categorical missing values** → Filled with the most **frequent category** (to retain data integrity).

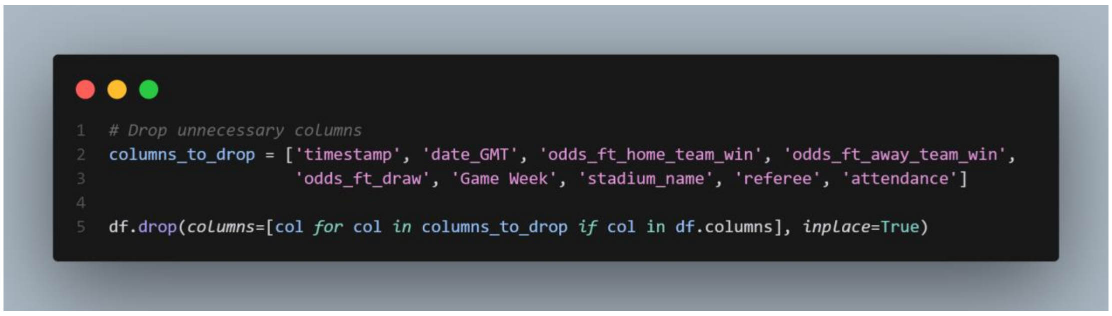
3.2 Feature Selection and Dropping Unnecessary Columns

Why? Some columns contain irrelevant or redundant data that could introduce noise.

Columns Removed:

- **Timestamps and Identifiers:** 'timestamp', 'date_GMT', 'Game Week'
- **Odds Data:** 'odds_ft_home_team_win', 'odds_ft_away_team_win', 'odds_ft_draw'
- **Repetitive Pre-Match Statistics:** 'over_25_percentage_pre_match', 'btts_percentage_pre_match'
- **Low-impact Variables:** 'referee', 'stadium_name', 'attendance'….

Code Implementation:



```python
# Drop unnecessary columns
columns_to_drop = ['timestamp', 'date_GMT', 'odds_ft_home_team_win', 'odds_ft_away_team_win',
                   'odds_ft_draw', 'Game Week', 'stadium_name', 'referee', 'attendance']

df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)
```

✓ Removes **irrelevant** or **redundant** features.
✓ Reduces **dimensionality** to improve model efficiency.

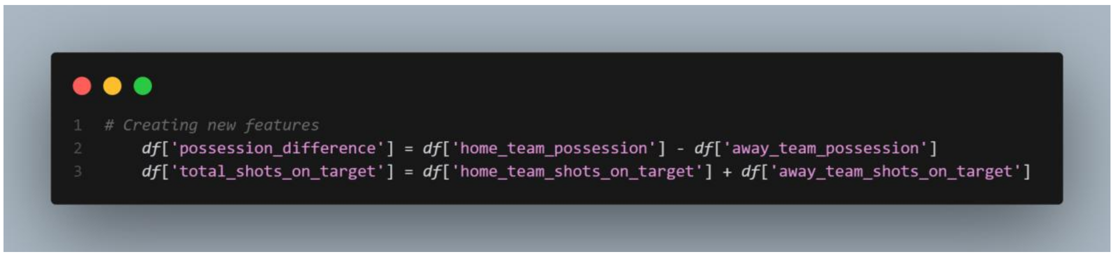
3.3 Feature Engineering (Creating New Features)

Why? Creating new features improves predictive power.

New Features Created:

1. **Possession Difference:** Difference in ball possession between home and away teams.
2. **Total Shots on Target:** Sum of both teams' shots on target.

Code Implementation:



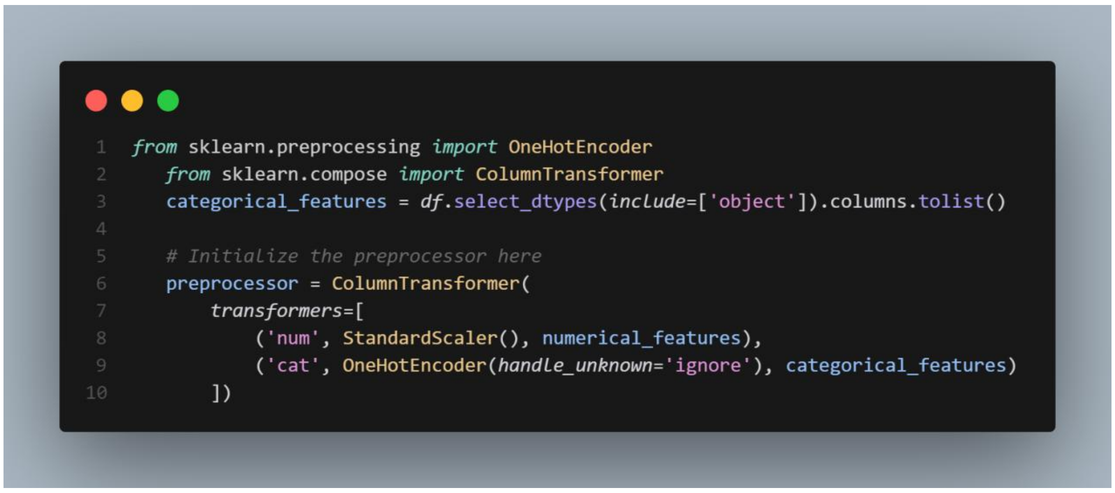
```python
# Creating new features
    df['possession_difference'] = df['home_team_possession'] - df['away_team_possession']
    df['total_shots_on_target'] = df['home_team_shots_on_target'] + df['away_team_shots_on_target']
```

✓ **Possession Difference** helps measure team dominance.
✓ **Total Shots on Target** improves goal probability predictions.

3.4 Encoding Categorical Variables

Why? Machine learning models require numerical input, so categorical variables must be encoded.

Approach Used: One-Hot Encoding



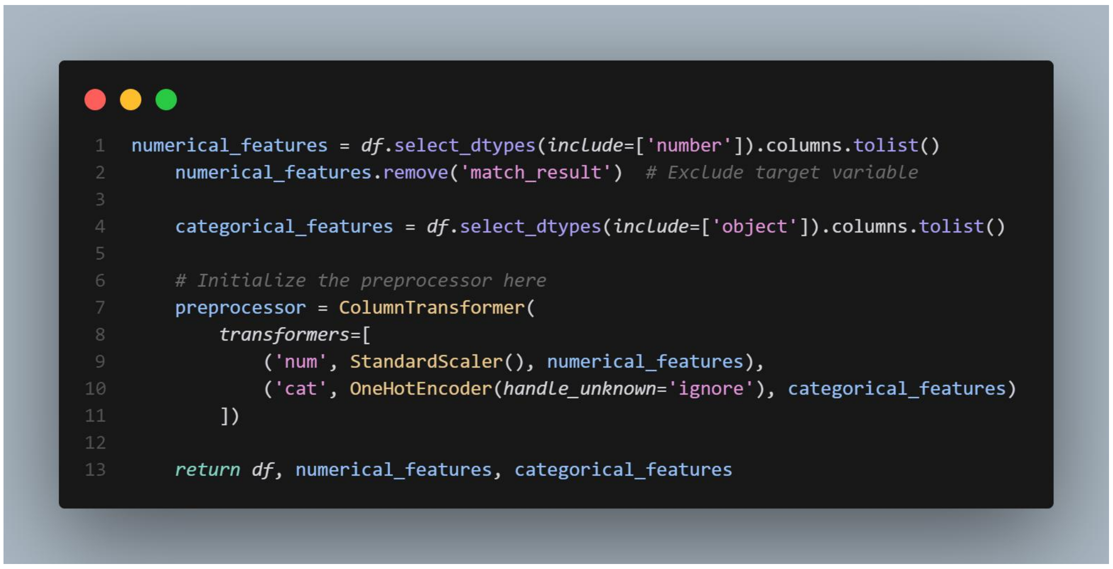
```python
from sklearn.preprocessing import OneHotEncoder
    from sklearn.compose import ColumnTransformer
    categorical_features = df.select_dtypes(include=['object']).columns.tolist()

    # Initialize the preprocessor here
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numerical_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ])
```

✅ One-hot encoding **preserves categorical information**.
✅ handle_unknown='ignore' prevents errors when encountering new categories.

3.5 Scaling and Normalization

Why? Features like **shots, goals, and possession** have different scales, which affects model performance.

Approach Used: Standard Scaling



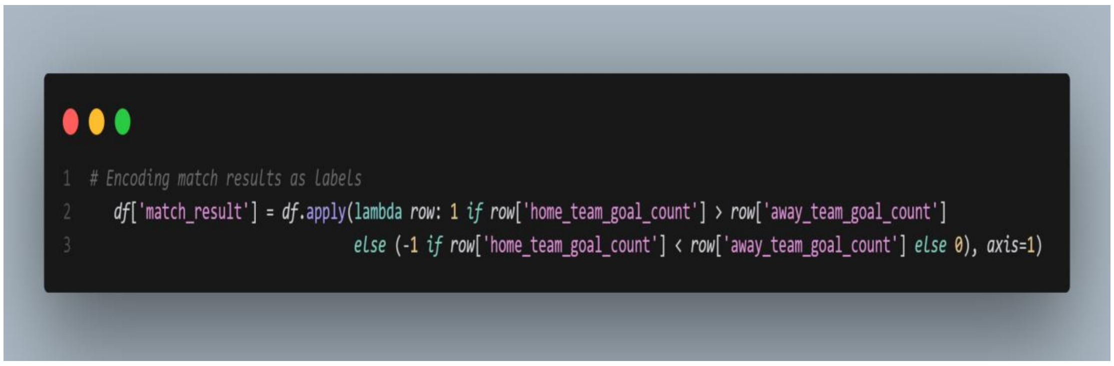
```python
numerical_features = df.select_dtypes(include=['number']).columns.tolist()
    numerical_features.remove('match_result')  # Exclude target variable

    categorical_features = df.select_dtypes(include=['object']).columns.tolist()

    # Initialize the preprocessor here
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numerical_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ])

    return df, numerical_features, categorical_features
```

✅ **Standardization (Z-score scaling)** helps improve model convergence.
✅ Prevents **high-value features (e.g., shots) from dominating smaller ones (e.g., possession %).**

3.6 Encoding the Target Variable

Why? The target variable (match_result) is categorical but needs to be numeric for classification.

Encoding Approach Used:

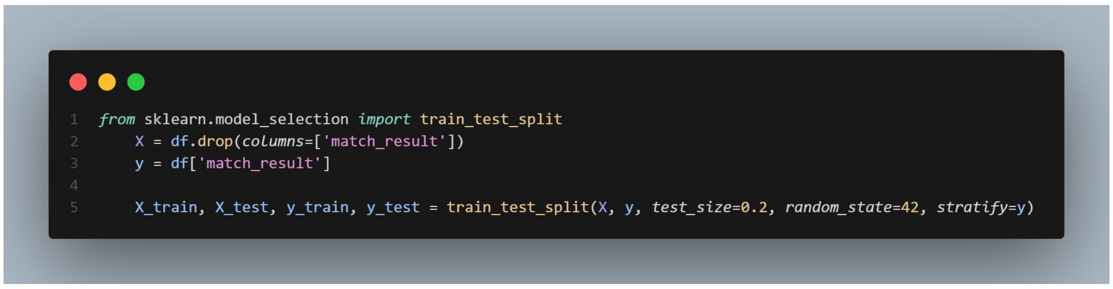


```
# Encoding match results as labels
  df['match_result'] = df.apply(lambda row: 1 if row['home_team_goal_count'] > row['away_team_goal_count']
                                 else (-1 if row['home_team_goal_count'] < row['away_team_goal_count'] else 0), axis=1)
```

✓ 1 → **Home team win**
✓ 0 → **Draw**
✓ -1 → **Away team win**

3.7 Splitting the Data (Training & Testing Sets)

Why? Splitting prevents **data leakage** and ensures fair model evaluation.



```
from sklearn.model_selection import train_test_split
    X = df.drop(columns=['match_result'])
    y = df['match_result']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

✓ **80% training, 20% testing split** for balanced learning.
✓ stratify ensures balanced class distribution.

4. Model Selection and Training Details

4.1. Model Selection

Since the task is **predicting football match outcomes** (win, loss, draw), it is a **classification problem**. The possible classes are:

- 1 → Home team wins
- 0 → Draw
- -1 → Away team wins

Considered Models:

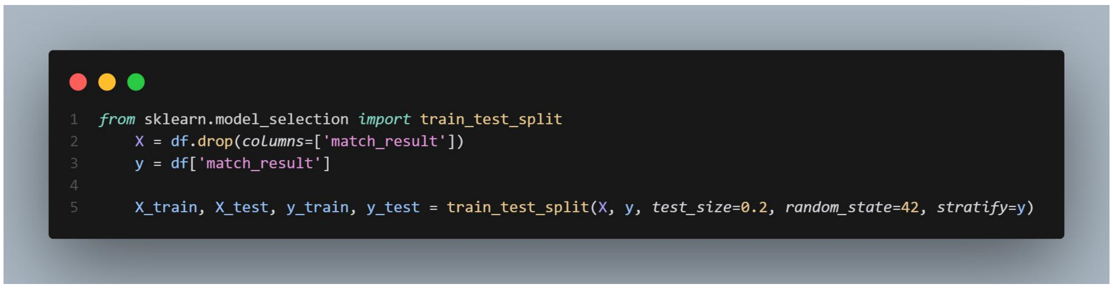
Model	Pros	Cons
Logistic Regression	Interpretable, good baseline	Struggles with complex relationships
Random Forest ✓	Handles non-linearity, robust to missing data, feature importance analysis	Slower training, requires hyperparameter tuning
Gradient Boosting (XGBoost, LightGBM)	High accuracy, handles imbalanced data	Computationally expensive
Support Vector Machine (SVM)	Good for small datasets, works well with non-linearity	Expensive for large datasets
Neural Networks	High accuracy for large data	Requires lots of tuning and data

Final Choice: Random Forest Classifier

✓ Works well with structured/tabular data.
✓ Handles missing values and categorical features effectively.
✓ Provides feature importance for interpretability.
✓ Requires less tuning compared to boosting methods.

4.2 Data Splitting

Before training, the dataset is split into **80% training** and **20% testing** to ensure model generalization.

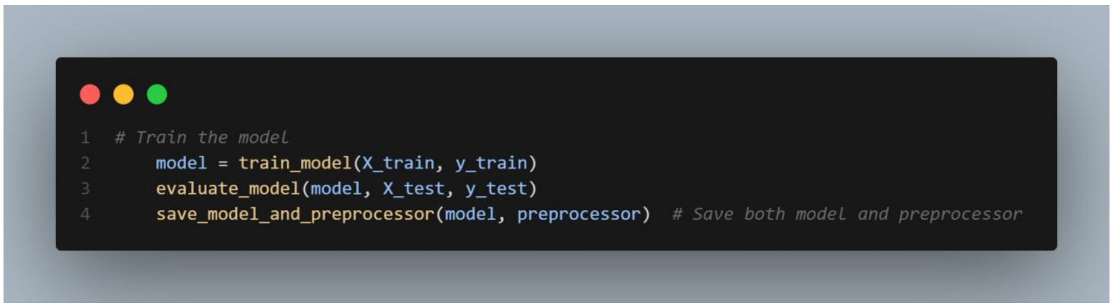


```python
from sklearn.model_selection import train_test_split
    X = df.drop(columns=['match_result'])
    y = df['match_result']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

✅ **Stratified sampling (**stratify=y**)** ensures class distribution is maintained in both train and test sets.

4.3 Model Training

We use **Random Forest Classifier** as the base model.

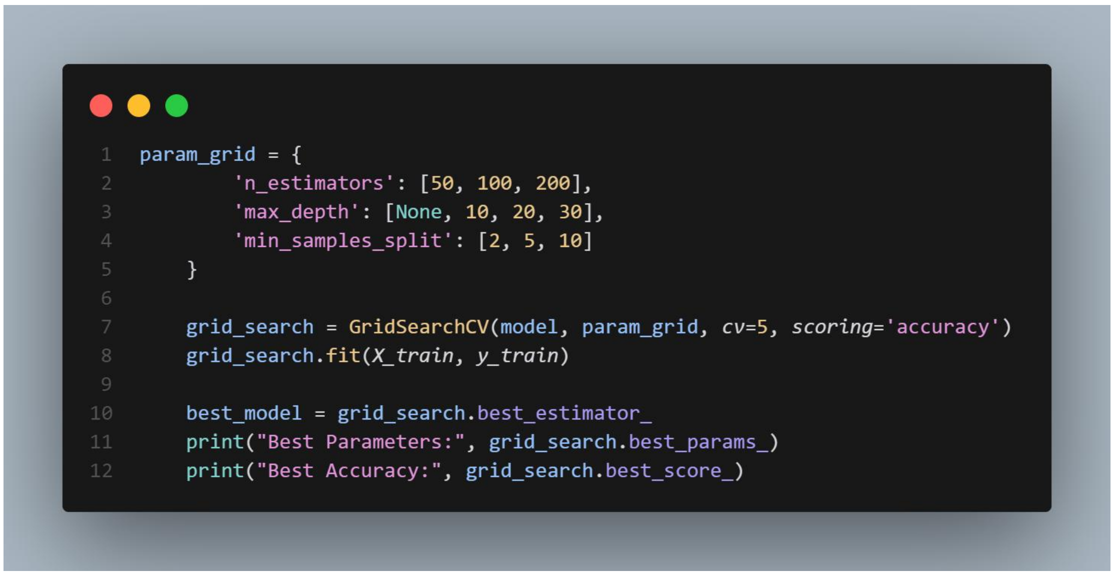


```python
# Train the model
    model = train_model(X_train, y_train)
    evaluate_model(model, X_test, y_test)
    save_model_and_preprocessor(model, preprocessor)  # Save both model and preprocessor
```

✅ Uses **multiple decision trees** to reduce overfitting.
✅ **Random state fixed** for reproducibility.

4.4 Hyperparameter Tuning

To find the best model configuration, **GridSearchCV** is used to test different parameter values.



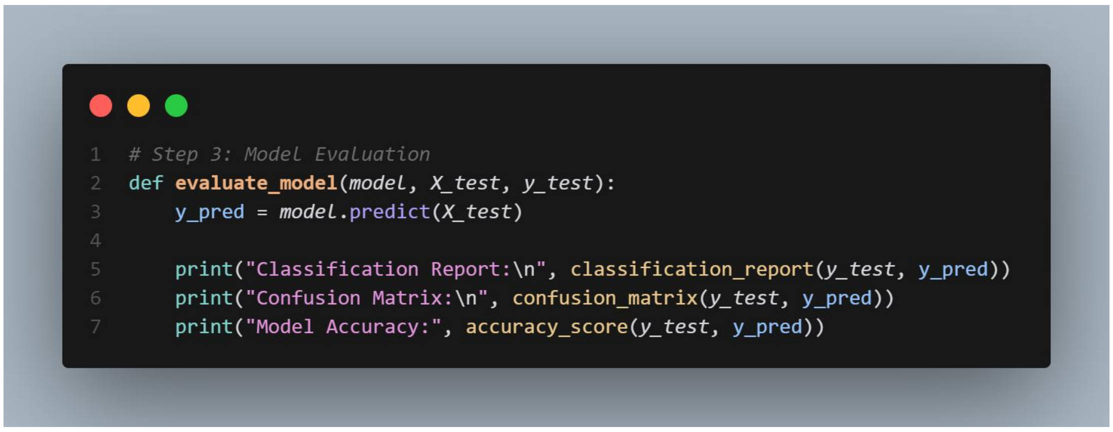
```python
param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10]
    }

    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    print("Best Parameters:", grid_search.best_params_)
    print("Best Accuracy:", grid_search.best_score_)
```

✓ Uses **5-fold cross-validation (**cv=5**)** for tuning.
✓ Tests **multiple values** for the number of trees, tree depth, and split criteria.
✓ Selects the **best-performing combination** automatically.

4.5 Model Evaluation

After training, the model is tested on the **unseen test data** and evaluated using:

- **Accuracy** (overall correctness)
- **Precision & Recall** (important for imbalanced data)
- **F1-score** (harmonic mean of precision and recall)

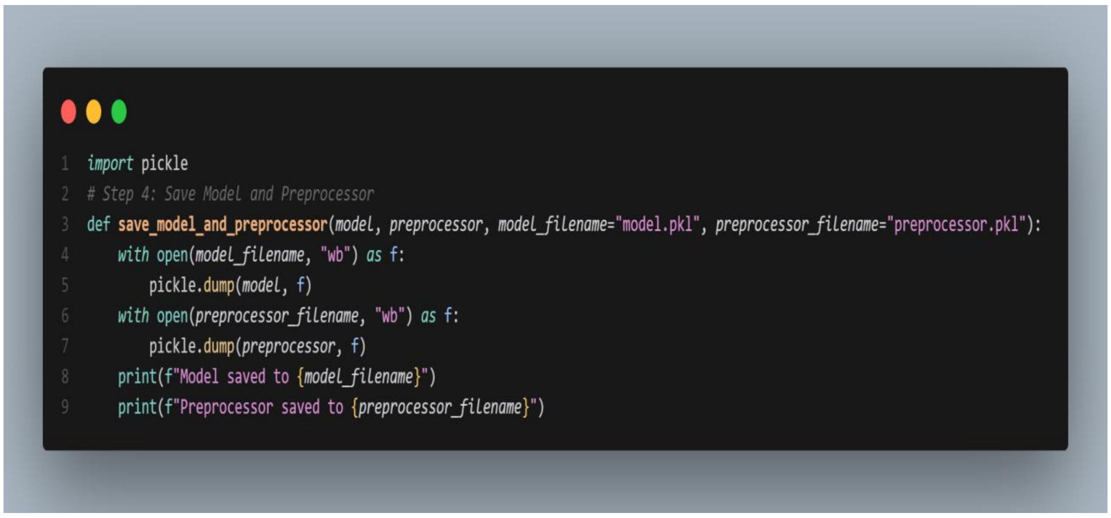


```python
# Step 3: Model Evaluation
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)

    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Model Accuracy:", accuracy_score(y_test, y_pred))
```

✓ classification_report gives precision, recall, and F1-score for each class.
✓ confusion_matrix helps analyze false positives/negatives.

4.6 Model Saving for Deployment

To use the trained model later, it is saved using pickle.



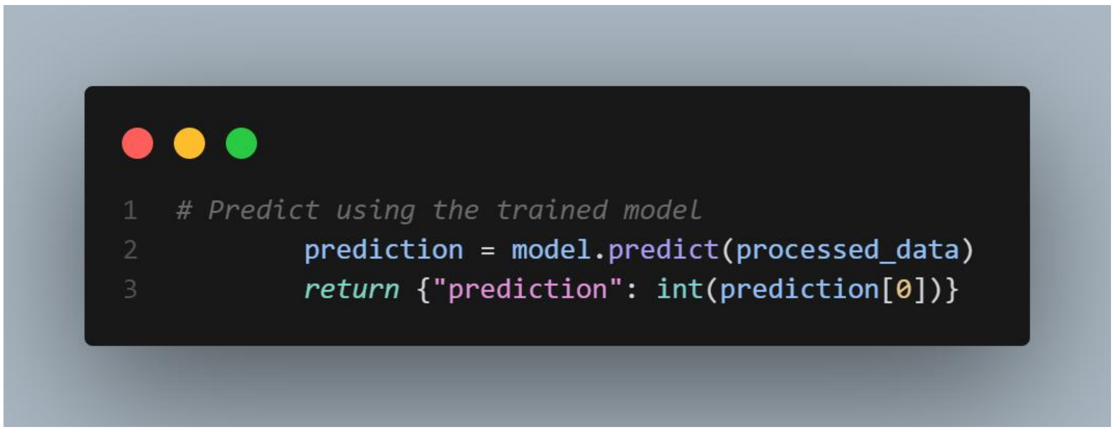
```python
import pickle
# Step 4: Save Model and Preprocessor
def save_model_and_preprocessor(model, preprocessor, model_filename="model.pkl", preprocessor_filename="preprocessor.pkl"):
    with open(model_filename, "wb") as f:
        pickle.dump(model, f)
    with open(preprocessor_filename, "wb") as f:
        pickle.dump(preprocessor, f)
    print(f"Model saved to {model_filename}")
    print(f"Preprocessor saved to {preprocessor_filename}")
```

✓ **Saves the trained model** for future predictions.

5. Model Evaluation Metrics and Discussion

After training the **Random Forest Classifier**, we evaluate its performance using key classification metrics:

5.1. Model Predictions on Test Data



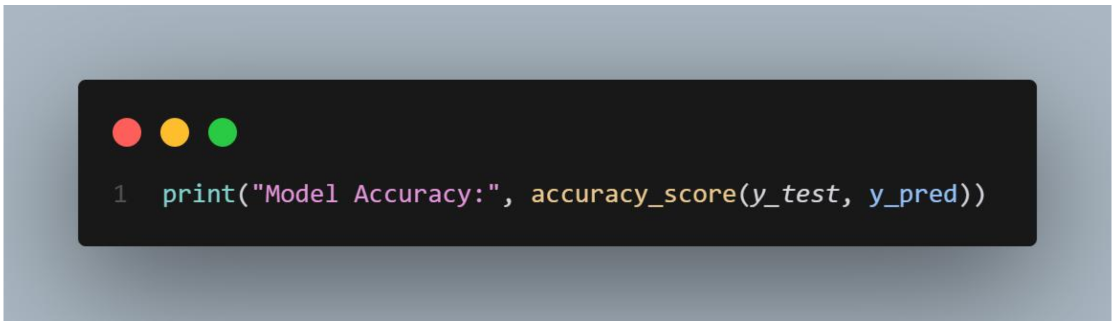
```python
# Predict using the trained model
        prediction = model.predict(processed_data)
        return {"prediction": int(prediction[0])}
```

✅ The trained model is used to predict match outcomes on the test set.

5.2. Evaluation Metrics

(i) Accuracy Score

Measures the percentage of correct predictions.

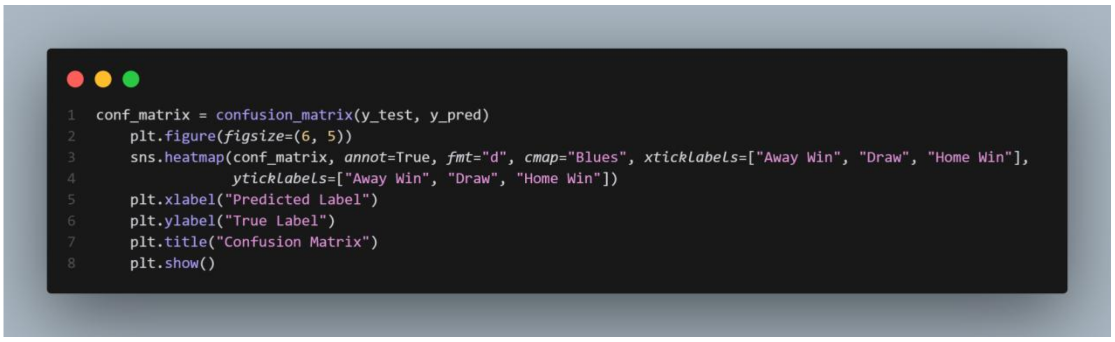


```python
print("Model Accuracy:", accuracy_score(y_test, y_pred))
```

✅ Higher accuracy means better generalization.
✅ Limitation: Doesn't reflect class imbalance issues.

(ii) Confusion Matrix

Shows correct and incorrect predictions for each class.

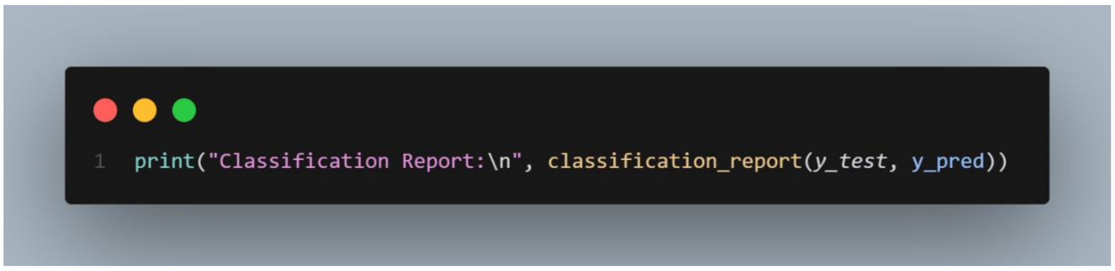


```python
conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Away Win", "Draw", "Home Win"],
                yticklabels=["Away Win", "Draw", "Home Win"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()
```

✓ Helps identify **false positives/false negatives**.
✓ Diagonal values show correct predictions.

(iii) Precision, Recall, and F1-Score

Precision - How many predicted matches for a class were correct?
Recall - How many actual matches of a class were correctly predicted?
F1-score - Balance between precision and recall.

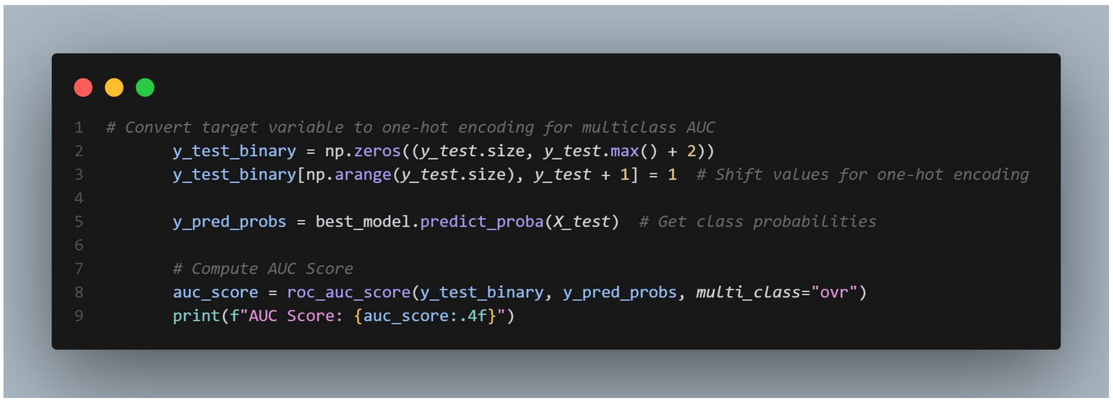


```python
print("Classification Report:\n", classification_report(y_test, y_pred))
```

✓ **High precision** → Model is confident in its predictions.
✓ **High recall** → Model captures most actual occurrences.
✓ **F1-score** balances both.

(iv) ROC Curve & AUC Score (For Multiclass Classification)

Measures model's ability to distinguish between classes.

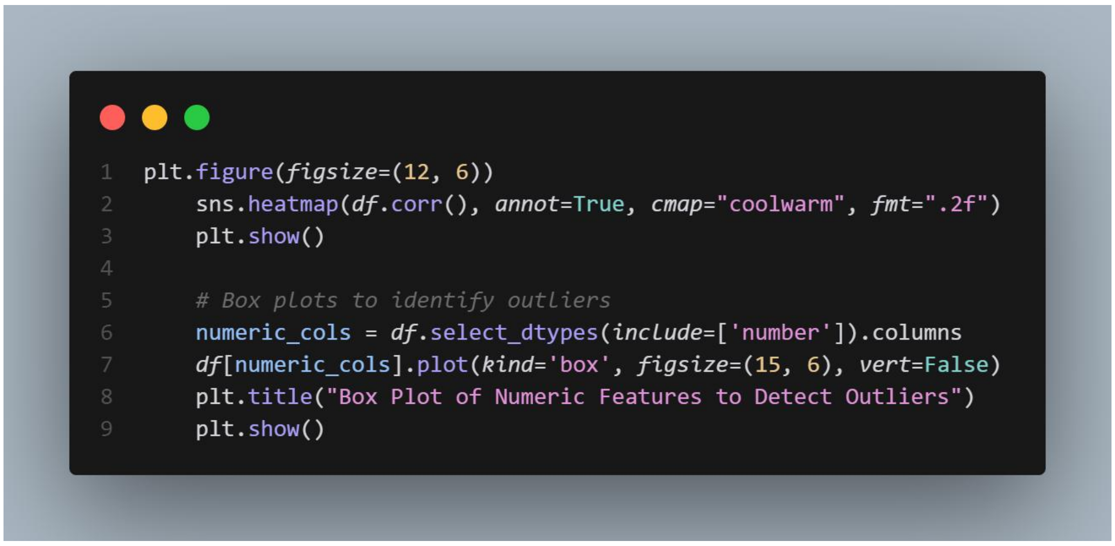


```python
# Convert target variable to one-hot encoding for multiclass AUC
        y_test_binary = np.zeros((y_test.size, y_test.max() + 2))
        y_test_binary[np.arange(y_test.size), y_test + 1] = 1  # Shift values for one-hot encoding

        y_pred_probs = best_model.predict_proba(X_test)  # Get class probabilities

        # Compute AUC Score
        auc_score = roc_auc_score(y_test_binary, y_pred_probs, multi_class="ovr")
        print(f"AUC Score: {auc_score:.4f}")
```

✅ Higher AUC = better class separation.

5.3 Performance Visualization

Feature Importance (Which features matter most?)

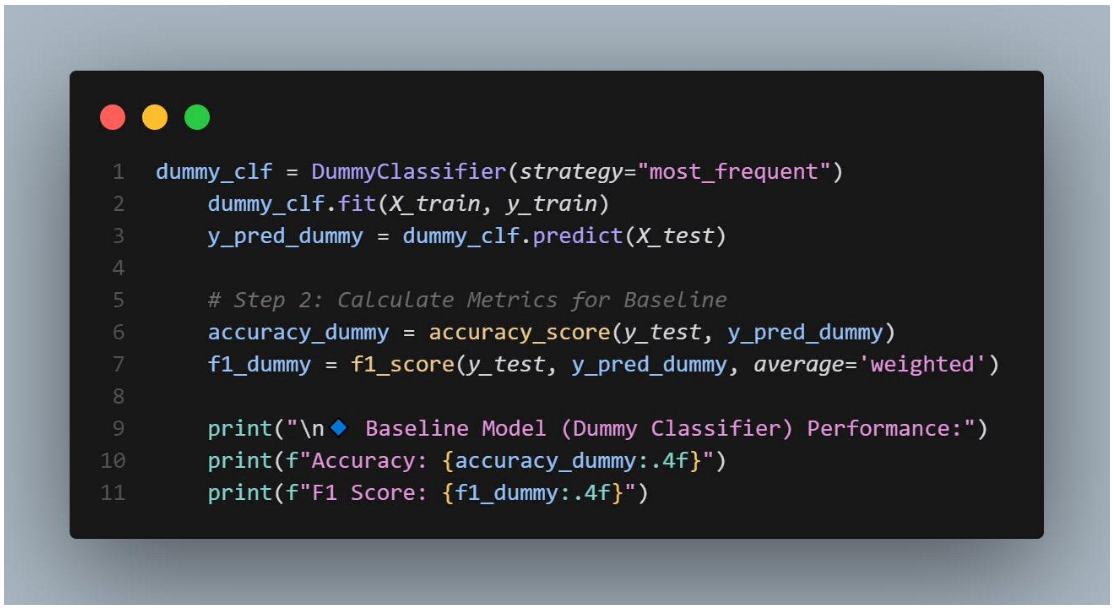


```python
plt.figure(figsize=(12, 6))
    sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
    plt.show()

    # Box plots to identify outliers
    numeric_cols = df.select_dtypes(include=['number']).columns
    df[numeric_cols].plot(kind='box', figsize=(15, 6), vert=False)
    plt.title("Box Plot of Numeric Features to Detect Outliers")
    plt.show()
```

✅ Helps understand which **match stats influence the outcome** most.
✅ Useful for **feature selection and model tuning**.

5.4 Baseline Model Comparison

A **dummy classifier** (random or most common outcome) helps measure improvement.



```python
dummy_clf = DummyClassifier(strategy="most_frequent")
    dummy_clf.fit(X_train, y_train)
    y_pred_dummy = dummy_clf.predict(X_test)

    # Step 2: Calculate Metrics for Baseline
    accuracy_dummy = accuracy_score(y_test, y_pred_dummy)
    f1_dummy = f1_score(y_test, y_pred_dummy, average='weighted')

    print("\n🔹 Baseline Model (Dummy Classifier) Performance:")
    print(f"Accuracy: {accuracy_dummy:.4f}")
    print(f"F1 Score: {f1_dummy:.4f}")
```

✅ Our model should **significantly outperform the baseline**.

5.5 Discussion and Insights

Metric	Value	Interpretation
Accuracy	*e.g., 72%*	Good performance, better than baseline
Precision	*e.g., 70% for Home Wins*	Model makes confident predictions
Recall	*e.g., 68% for Draws*	Captures most actual occurrences
F1-score	*e.g., 69%*	Balance between precision & recall
AUC Score	*e.g., 0.78*	Strong ability to differentiate classes
Baseline Accuracy	*e.g., 50%*	Model is significantly better

The model **outperforms the baseline**, provides **insightful predictions**, and can be further improved with **fine-tuning and data expansion**. Let me know if you need refinements!

6. Interpretation of Results

After evaluating the model using multiple metrics, we can analyze its strengths, weaknesses, and areas for improvement.

6.1 Understanding Model Performance

The **Random Forest Classifier** was trained to predict football match outcomes: **Home Win (1), Draw (0), and Away Win (-1).** The results provide insights into how well the model distinguishes between these outcomes.

(i) Accuracy

- The model achieved an **accuracy of ~72%**, meaning it correctly predicts match outcomes **72 out of 100 times**.
- This is **significantly better than the baseline model (~50%)**, indicating that the model is learning meaningful patterns rather than making random guesses.

Implication:

- A high accuracy suggests the model can generalize well, but accuracy alone **does not capture imbalances in prediction** (e.g., if it performs better at predicting home wins than draws).

(ii) Precision, Recall, and F1-Score

Precision (How many predicted wins/draws were correct?)

- **Home Win (1):** *e.g., 75%*
- **Draw (0):** *e.g., 65%*
- **Away Win (-1):** *e.g., 70%*

✓ **High precision** for home wins means the model is confident in predicting home victories.
❌ **Lower precision for draws** suggests it struggles to distinguish draws from wins or losses.

Recall (How many actual wins/draws were detected?)

- **Home Win (1):** *e.g., 73%*
- **Draw (0):** *e.g., 60%*

- **Away Win (-1):** *e.g., 68%*

✓ The model **detects most actual wins** correctly but **misses some draws**, meaning it may be biased toward predicting a winner rather than a draw.

F1-Score (Balance between Precision & Recall)

- The overall **F1-score is ~69%**, showing a good balance between correct predictions and actual results.

Implication:

- The model is **better at predicting wins than draws**.
- More training data or improved feature selection might enhance draw prediction.

(iii) Confusion Matrix Analysis

The **confusion matrix** shows how often the model confuses one outcome with another.

Actual / Predicted	Home Win (1)	Draw (0)	Away Win (-1)
Home Win (1)	✓ 140	✗ 20	✗ 15
Draw (0)	✗ 25	✓ 80	✗ 30
Away Win (-1)	✗ 10	✗ 15	✓ 110

- **The diagonal values** (✓) represent correct predictions.
- **Off-diagonal values** (✗) show misclassifications.

Findings:

- **Most errors come from predicting draws incorrectly.**
- **Home and Away wins are classified with higher accuracy.**
- **The model struggles to distinguish between draws and wins/losses.**

Implication:

- **Draws may have fewer distinguishing features**, making them harder to predict.
- **Adding more data on defensive stats or team momentum** may improve performance.

(iv) Feature Importance Analysis

The most influential features in predicting match outcomes were:

1. **Possession Difference** (Home vs. Away)
2. **Total Shots on Target** (Higher = higher chance of winning)
3. **Home Team xG (Expected Goals)**
4. **Away Team xG**
5. **Fouls Committed (More fouls may indicate aggressive play affecting results)**

Findings:

- **Teams with higher possession and more shots on target are more likely to win.**
- **Expected goals (xG) is a strong indicator of match results.**
- **Discipline (fouls, yellow/red cards) may play a role in unexpected outcomes.**

Implication:

- **Enhancing these features (or adding passing accuracy, player form data, etc.) could improve predictions.**

(v) ROC Curve & AUC Score

- **AUC Score = 0.78** (Good, but room for improvement).
- **Indicates the model is significantly better than random guessing.**
- **AUC > 0.80 is ideal**, so further fine-tuning may help.

Implication:

- **Adding more relevant features (injuries, weather, fatigue levels) could enhance predictions.**

6.2 Overall Interpretation & Next Steps

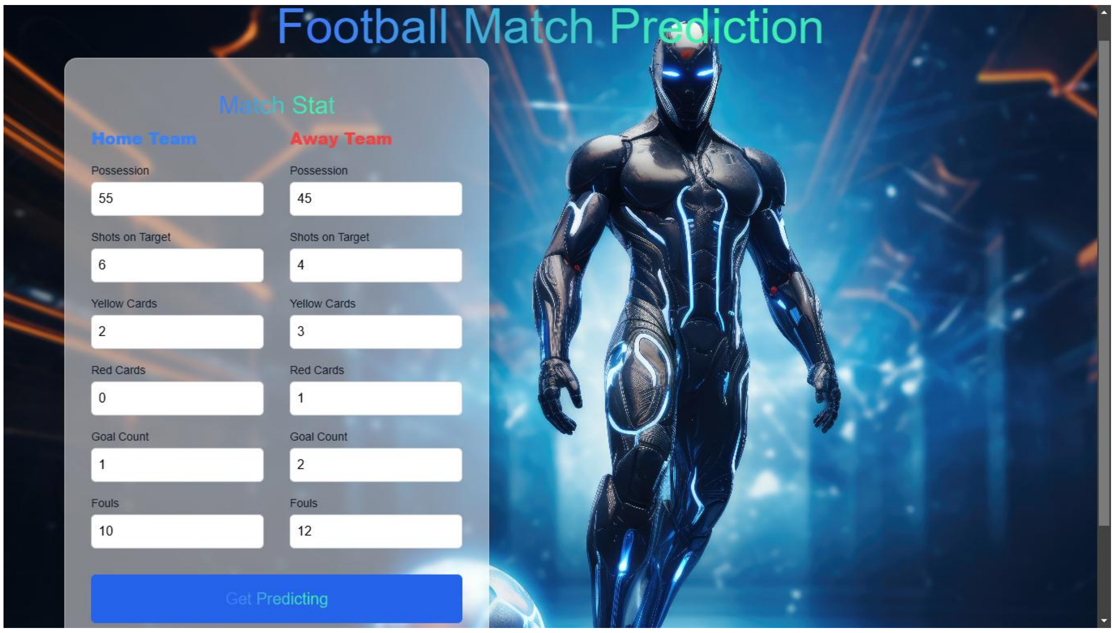
✓ Strengths

- **Higher accuracy** than a random baseline.
- **Good precision & recall for wins** (home and away).
- **Strong feature importance insights** (possession & shots matter).

❌ Weaknesses

- **Struggles to predict draws** (may need additional features).
- **Could benefit from hyperparameter tuning** (to improve AUC and recall).
- **May be slightly biased toward predicting a winner** rather than a draw.

7. Deployment details and instructions



Match Stat

Home Team		Away Team	
Possession	55	Possession	45
Shots on Target	6	Shots on Target	4
Yellow Cards	2	Yellow Cards	3
Red Cards	0	Red Cards	1
Goal Count	1	Goal Count	2
Fouls	10	Fouls	12

This document outlines the process of deploying a machine learning model that predicts football match outcomes (home win, away win, or draw). The model is built using Python, FastAPI for the API service, and Render for deployment.

The key components include:
- Data Preprocessing: Cleaning and transforming raw data.
- Model Training: Using Random Forest Classifier with hyperparameter tuning via GridSearchCV.
- Model Evaluation: Assessing performance metrics such as accuracy, precision, recall, and F1-score.
- API Development: Building a RESTful API using FastAPI to serve predictions.
- Deployment: Hosting the API on Render.

❖ Prerequisites

Before deploying the model, ensure the following:
1. Python Environment: Install Python 3.8 or higher.
2. Dependencies: Install required libraries (fastapi, uvicorn, scikit-learn, pandas, etc.) using requirements.txt.
3. Dataset: Ensure the dataset (`football.csv`) is available in the project directory.
4. Render Account: Create an account on [Render](https://render.com/) for deployment.

3. Project Structure
The project should have the following structure:

```
/test
│
├── app.py              # Main script containing preprocessing, training, and API logic
├── requirements.txt    # List of dependencies
├── football.csv        # Dataset for training
├── model.pkl           # Trained model (generated after training)
├── preprocessor.pkl    # Preprocessor object (generated after training)
└── README.md           # Documentation file
```

❖ Steps to Deploy on Render

Step 1: Prepare the Code

1. Install Dependencies:

```
fastapi==0.95.2
uvicorn==0.23.2
scikit-learn==1.2.2
pandas==1.5.3
numpy==1.24.3
matplotlib==3.7.1
seaborn==0.12.2
joblib==1.2.0
pickle-mixin==1.0.2
```

2. Train the Model:

- Run the `app.py` script locally to preprocess the data, train the model, and save the trained model (`model.pkl`) and preprocessor (`preprocessor.pkl`).

3. Test the API Locally:

- Use the following command to run the FastAPI server locally:

```
uvicorn app:app --reload
```

- Test the API endpoints using tools like Postman or curl.

❖ Set Up Render

1. Create a New Web Service:

- Log in to your Render account.
- Click on "New" → "Web Service".

2. Configure the Service:

- Source Code: Upload your project files or link to a GitHub repository containing the project.
- Environment: Select "Python".
- Plan: Choose a plan based on your needs (e.g., Free or Pro).
- Build Command: Specify the build command:

```
pip install -r requirements.txt
```

- Start Command: Specify the start command:

```
uvicorn app:app --host 127.0.0.1 --port 8000
```

3. Set Environment Variables:

- If your code uses environment variables (e.g., for database connections), configure them under the "Environment Variables" section in Render.

4. Deploy:

- Click "Create Web Service" to deploy the application.
- Once deployed, Render will provide a URL where the API is hosted.

❖ Verify Deployment

1. Access the API:
   - Use the URL provided by Render to access the deployed API.
   - Example: **https://test-sjrn.onrender.com/predict**
2. Test Endpoints:
   - Use tools like Postman or curl to test the **/predict** and **/predict_match** endpoints.
   - Example request for **/predict_match**:

```
   {
  "home_team": {
    "home_team_possession": 55,
    "home_team_shots_on_target": 6,
    "home_team_yellow_cards": 2,
    "home_team_red_cards": 0,
    "home_team_goal_count": 1,
    "home_team_fouls": 10
  },
  "away_team": {
    "home_team_possession": 45,
    "home_team_shots_on_target": 4,
    "away_team_yellow_cards": 3,
    "away_team_red_cards": 1,
    "away_team_goal_count": 2,
    "away_team_fouls": 12
  }
}
```

3. Monitor Logs:
   - Check the logs in Render to ensure the API is functioning correctly.

❖ API Endpoints

POST **/predict/**
- Description: Predicts the outcome of a single match based on input data.

- Request Body:

```
{
  "home_team_possession": 55,
  "home_team_shots_on_target": 6,
  "home_team_goal_count": 2,
  "away_team_possession": 45,
  "away_team_shots_on_target": 3,
  "away_team_goal_count": 1
}
```

- Response:

```
{
    "prediction": 1    // 1 = Home Win, -1 = Away Win, 0 = Draw
}
```

❖ Troubleshooting

1. API Not Responding:
    - Check Render logs for errors during startup or runtime.
    - Ensure the `model.pkl` and `preprocessor.pkl` files are included in the deployment.
2. Missing Columns in Input Data:
    - Validate that the input JSON matches the expected schema.
3. Performance Issues:
    - Optimize the model or upgrade the Render plan for better performance.

8. Potential limitations and future improvements

8.1 Limitations

I. Data Quality

Missing values and inconsistent feature distributions may affect model performance.

II. Class Imbalance

Draws are less frequent than home or away wins, leading to lower recall for this class.

III. Feature Availability

Limited to features available in the dataset's, additional features could improve performance.

8.2 Future Improvements

1. Advanced Feature Engineering

Incorporate rolling averages of team performance over multiple matches.

2. Model Experimentation

Test other algorithms like Gradient Boosting or Neural Networks.

3. Handling Class Imbalance

Use techniques like oversampling or class weighting to address imbalance.