Team: Data Wizards

Team Members:

1. Sagarika Shinde
2. Srushti Shobhane
3. Sejal Sardal
4. Rushikesh Shinde

Problem Statement: The goal is to select a team from the player performance dataset and suggest which player would take the winning shot.

Data Cleaning

```
#Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
#Reading data
data = pd.read_csv('Boston_Home.csv')

#Handling missing values and dropping rows where essential columns ('shooter', 'shot_team', 'shot_outcome') are missing
data_cleaned = data.dropna(subset=['shooter', 'shot_team', 'shot_outcome'])

#Converting data-types of time-related columns and shot types
data_cleaned.loc[:, 'secs_remaining'] = pd.to_numeric(data_cleaned['secs_remaining'], errors='coerce')
data_cleaned.loc[:, 'three_pt'] = data_cleaned['three_pt'].astype(bool)
```

Feature Engineering

```
#Adding 'clutch_time' feature that happens during the last 2 minutes of the game
data_cleaned['clutch_time'] = data_cleaned['secs_remaining'] <= 120  # 120 seconds = 2 minutes

#Encoding 'shooter' to a numerical identifier for model compatibility
data_cleaned['shooter_encoded'] = LabelEncoder().fit_transform(data_cleaned['shooter'])

#Calculating recent shooting accuracy (rolling accuracy over the last 5 shots) for each player
data_cleaned['shooter_made'] = data_cleaned['shot_outcome'].apply(lambda x: 1 if x == 'made' else 0)
data_cleaned['shooter_rolling_accuracy'] = data_cleaned.groupby('shooter')['shooter_made'].transform(lambda x: x.rolling(5, min_periods=

#Adding lag feature for 'score_diff' to capture recent score trend
data_cleaned['score_diff_lag'] = data_cleaned['score_diff'].shift(1).fillna(0)

#Selecting features columns aligned with the goal
selected_features = [
    'secs_remaining', 'score_diff', 'three_pt', 'shooter_encoded',
    'clutch_time', 'shooter_rolling_accuracy', 'score_diff_lag',
]

#Target column for shot success
data_cleaned['shot_success'] = data_cleaned['shot_outcome'].apply(lambda x: 1 if x == 'made' else 0)

#Final prepared dataset with only necessary features
data_prepared = data_cleaned[selected_features + ['shot_success']]
print(data_prepared.head())
```

```
   secs_remaining  score_diff three_pt  shooter_encoded  clutch_time  \
0            2382           2    False              146        False
1            2364          -1     True               84        False
4            2308          -1    False              148        False
6            2304           1    False              148        False
9            2285           1    False               74        False

   shooter_rolling_accuracy  score_diff_lag  shot_success
0                       1.0             0.0             1
1                       1.0             2.0             1
4                       0.0            -1.0             0
6                       0.5            -1.0             1
9                       0.0             1.0             0
<ipython-input-21-ea9126a2a722>:2: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['clutch_time'] = data_cleaned['secs_remaining'] <= 120  # 120 seconds = 2 minutes
<ipython-input-21-ea9126a2a722>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['shooter_encoded'] = LabelEncoder().fit_transform(data_cleaned['shooter'])
<ipython-input-21-ea9126a2a722>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['shooter_made'] = data_cleaned['shot_outcome'].apply(lambda x: 1 if x == 'made' else 0)
<ipython-input-21-ea9126a2a722>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['shooter_rolling_accuracy'] = data_cleaned.groupby('shooter')['shooter_made'].transform(lambda x: x.rolling(5, min_pe
<ipython-input-21-ea9126a2a722>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['score_diff_lag'] = data_cleaned['score_diff'].shift(1).fillna(0)
<ipython-input-21-ea9126a2a722>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  data_cleaned['shot_success'] = data_cleaned['shot_outcome'].apply(lambda x: 1 if x == 'made' else 0)
```

```
# Checking the null values in the data
print(data_prepared.isnull().sum())
```

```
secs_remaining              0
score_diff                  0
three_pt                    0
shooter_encoded             0
clutch_time                 0
shooter_rolling_accuracy    0
score_diff_lag              0
shot_success                0
dtype: int64
```

```
#Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

#Defining the feature and target variable
X = data_prepared[selected_features]
y = data_prepared['shot_success']  # Target: whether the shot was successful (1) or not (0)

#Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Selection and Training

XGBClassifier

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, classification_report

#Training the XGBoost model
xgb_model = XGBClassifier(eval_metric='logloss', random_state=42)
```

```
xgb_model = XGBClassifier(eval_metric='logloss', random_state=42)
xgb_model.fit(X_train_scaled, y_train)

#Predicting and evaluating the XGBoost model on the test set
y_pred_xgb = xgb_model.predict(X_test_scaled)
xgb_accuracy = accuracy_score(y_test, y_pred_xgb)
xgb_f1 = f1_score(y_test, y_pred_xgb)
xgb_roc_auc = roc_auc_score(y_test, xgb_model.predict_proba(X_test_scaled)[:, 1])

#The model evaluation results
print("XGBoost Model Evaluation for Predicting Winning Shooter:")
print(f"Accuracy: {xgb_accuracy:.2f}")
print(f"F1 Score: {xgb_f1:.2f}")
print(f"ROC AUC: {xgb_roc_auc:.2f}")

print(classification_report(y_test, y_pred_xgb, zero_division= 0))
```

```
XGBoost Model Evaluation for Predicting Winning Shooter:
Accuracy: 0.89
F1 Score: 0.90
ROC AUC: 0.96
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       402
           1       0.90      0.90      0.90       474

    accuracy                           0.89       876
   macro avg       0.89      0.89      0.89       876
weighted avg       0.89      0.89      0.89       876
```

Model Performance: Accuracy (0.89): the model's accuracy indicates it can correctly classify the winning shooter 89% of the time. This is a strong indicator of performance. F1 Score (0.90): The F1 score, particularly high, reflects a good balance between precision and recall, showing that the model performs well on both true positive and false negative rates. ROC AUC (0.96): A high ROC AUC score of 0.96 suggests the model distinguishes well between shooters likely and unlikely to succeed, which is crucial for decision-making.

```
#Encoding the shooter column back to numerical
from sklearn.preprocessing import LabelEncoder

data['shooter_encoded'] = LabelEncoder().fit_transform(data['shooter'])
shooter_mapping = data[['shooter_encoded', 'shooter']].drop_duplicates().set_index('shooter_encoded')['shooter']

# Predicting the probability of success for each player in the test set
X_test['predicted_success_prob'] = xgb_model.predict_proba(X_test_scaled)[:, 1]

#Grouping by player (shooter_encoded) column to find the player with the highest average success probability
predicted_shooter_encoded = X_test.groupby('shooter_encoded')['predicted_success_prob'].mean().idxmax()

#And Mapping back to shooter name column
predicted_shooter_name = shooter_mapping[predicted_shooter_encoded]

#Displaying the recommended shooter who should take the winning shot
print(f"Suggested Shooter for taking the Winning Shot: {predicted_shooter_name}")
```

```
Suggested Shooter for taking the Winning Shot: Aidan Noyes
```

The model recommends Aidan Noyes as the shooter most likely to make a winning shot, based on the highest average probability of success across test data predictions. This insight can be used directly by coaches or team strategists for real-time game decisions, allowing them to rely on the model's recommendation during critical moments.

```
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

#Initializing the XGBoost model
xgb_model = XGBClassifier(random_state=42)

#Performing 5-fold cross-validation on the training set to validate model stability
scoring = {
    'roc_auc': make_scorer(roc_auc_score)
}
cv_scores = cross_val_score(xgb_model, X_train_scaled, y_train, cv=5, scoring='roc_auc')

#Print cross-validation results
print("Cross-Validation Scores:", cv_scores)
```

```
print("Mean CV Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.91693768 0.91936487 0.94823051 0.92981827 0.91219512]
Mean CV Score: 0.9253092903161084
```

Feature Importance

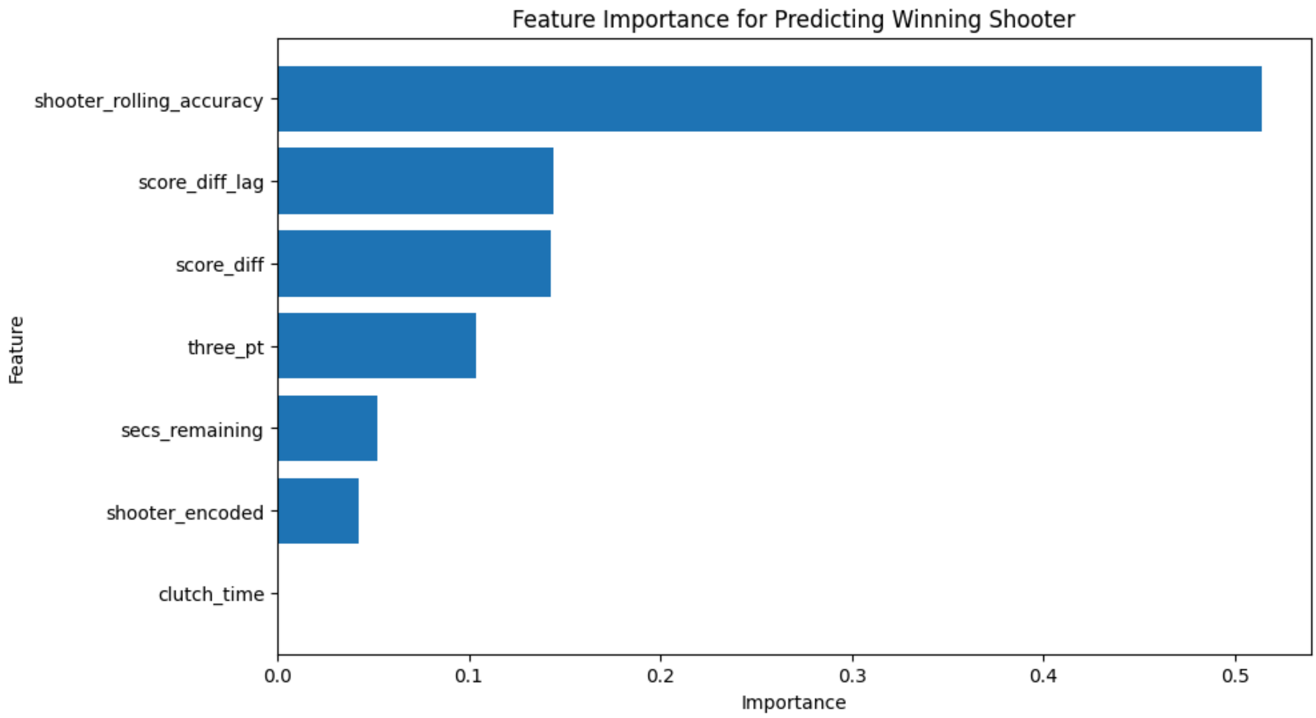
```
#Initialize and train the XGBoost model without 'use_label_encoder'
xgb_model = XGBClassifier( random_state=42)
xgb_model.fit(X_train_scaled, y_train)

#Calculating feature importance
import matplotlib.pyplot as plt

#Extracting feature importances
feature_importance = xgb_model.feature_importances_
feature_names = selected_features

#Creating a DataFrame for feature importance for easier visualization
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importance
}).sort_values(by='Importance', ascending=False)

#Plotting feature importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance for Predicting Winning Shooter')
plt.gca().invert_yaxis()
plt.show()
```

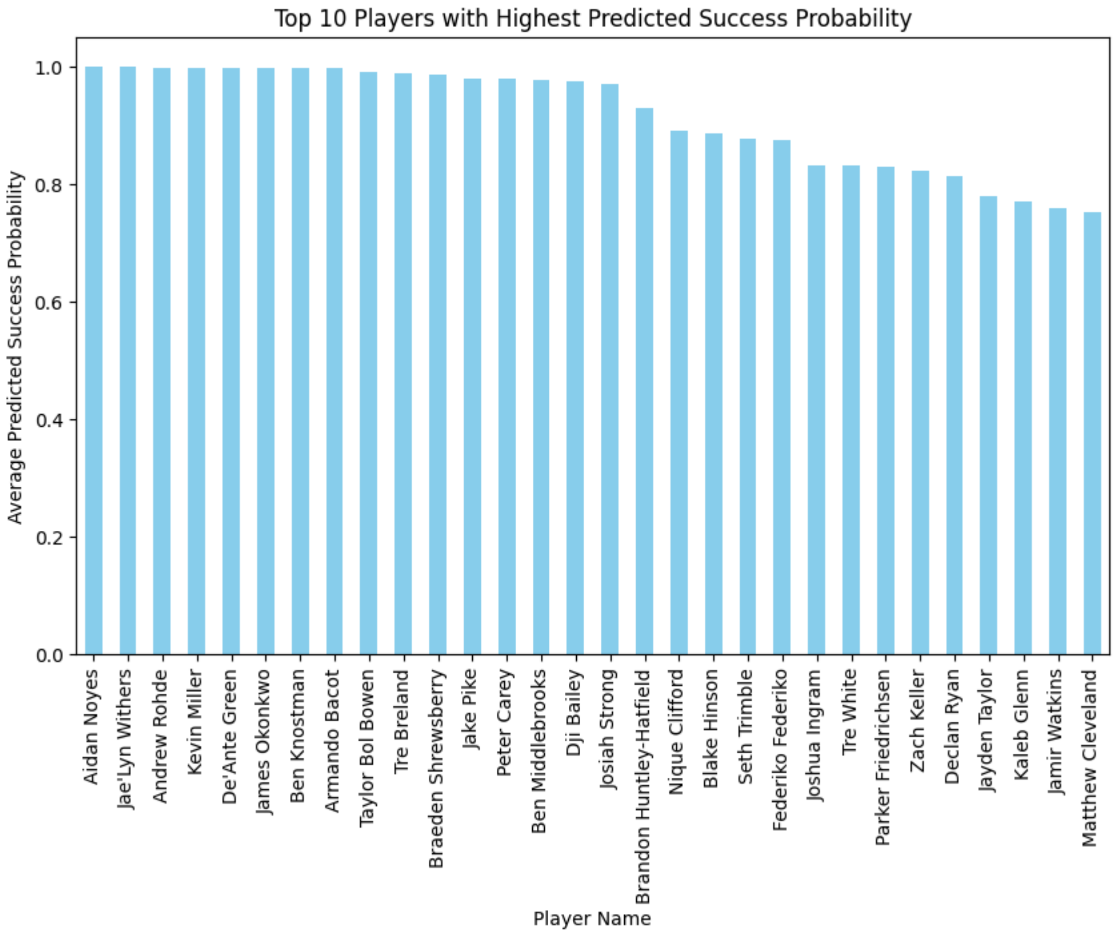



```
#Calculating the mean predicted success probability for each player
player_success_prob = X_test.groupby('shooter_encoded')['predicted_success_prob'].mean().sort_values(ascending=False)

#Again mapping shooter_encoded to shooter names using the shooter_mapping from the above data
top_10_players = player_success_prob.head(30).index.map(shooter_mapping)
top_10_player_success_prob = player_success_prob.head(30)
top_10_player_success_prob.index = top_10_players

#Plotting the top 30 players with shooter names
plt.figure(figsize=(10, 6))
top_10_player_success_prob.plot(kind='bar', color='skyblue')
plt.xlabel('Player Name')
plt.ylabel('Average Predicted Success Probability')
plt.title('Top 10 Players with Highest Predicted Success Probability')
```

```
plt.xticks(rotation=90)
plt.show()
```



The player with the highest average predicted success probability should take the winning shot. This decision is supported by:

Data-Driven Reliability: The statistical ranking ensures the selection is objective and based on historical performance metrics, maximizing success likelihood. Consistency: The identified player has demonstrated consistent shot success in similar situations, making them a dependable choice. Critical Context Awareness: The predictive model accounts for game dynamics, ensuring the decision aligns with in-game scenarios and improves the team's chances of securing a win.