Modelling

Import necessary libraries

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, roc_curve, r
import matplotlib.pyplot as plt
```

Load and visualize features and output variable

In [ ]:
```
# Load the data
X = pd.read_csv('datasets/raw_data_final/features_model.csv', index_col=0).drop(columm
y = pd.read_csv('datasets/raw_data_final/output_variable_model.csv', index_col=0)
```

In [ ]:
```
X.head(5)
```

Out[ ]:

	CLUB_NAME	CURRENT_INTERNATIONAL	AGE	MIN_PLAYING	DIST_STANDARD	DEF 3RD_TOUCHES
0	52	78	0.409091	0.584089	0.215017	0.837452
1	52	68	0.500000	0.547529	0.262799	0.584601
2	52	32	0.545455	0.539631	0.361775	0.517110
3	52	91	0.454545	0.668617	0.447099	0.774715
4	52	88	0.545455	0.290143	0.412969	0.318441

5 rows × 54 columns

In [ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2384 entries, 0 to 2383
Data columns (total 54 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   CLUB_NAME                              2384 non-null   int64  
 1   CURRENT_INTERNATIONAL                  2384 non-null   int64  
 2   AGE                                    2384 non-null   float64
 3   MIN_PLAYING                            2384 non-null   float64
 4   DIST_STANDARD                          2384 non-null   float64
 5   DEF 3RD_TOUCHES                        2384 non-null   float64
 6   ATT 3RD_TOUCHES                        2384 non-null   float64
 7   ATT PEN_TOUCHES                        2384 non-null   float64
 8   ATT_TAKE                               2384 non-null   float64
 9   TOTDIST_CARRIES                        2384 non-null   float64
 10  CPA_CARRIES                            2384 non-null   float64
 11  MIS_CARRIES                            2384 non-null   float64
 12  SUBS_SUBS                              2384 non-null   float64
 13  UNSUB_SUBS                             2384 non-null   float64
 14  PLUS_PER__MINUS__TEAM.SUCCESS          2384 non-null   float64
 15  TOTDIST_TOTAL                          2384 non-null   float64
 16  CRSPA                                  2384 non-null   float64
 17  FLS                                    2384 non-null   float64
 18  FLD                                    2384 non-null   float64
 19  OFF                                    2384 non-null   float64
 20  CRS                                    2384 non-null   float64
 21  WON_AERIAL                             2384 non-null   float64
 22  LOST_AERIAL                            2384 non-null   float64
 23  ATT 3RD_TACKLES                        2384 non-null   float64
 24  LOST_CHALLENGES                        2384 non-null   float64
 25  PASS_BLOCKS                            2384 non-null   float64
 26  TKL+INT                                2384 non-null   float64
 27  LEAGUE_COUNTRY_England                 2384 non-null   int64  
 28  LEAGUE_COUNTRY_France                  2384 non-null   int64  
 29  LEAGUE_COUNTRY_Germany                 2384 non-null   int64  
 30  LEAGUE_COUNTRY_Italy                   2384 non-null   int64  
 31  LEAGUE_COUNTRY_Spain                   2384 non-null   int64  
 32  POSITION_Attack - Centre-Forward       2384 non-null   int64  
 33  POSITION_Attack - Left Winger          2384 non-null   int64  
 34  POSITION_Attack - Right Winger         2384 non-null   int64  
 35  POSITION_Attack - Second Striker       2384 non-null   int64  
 36  POSITION_Defender - Centre-Back        2384 non-null   int64  
 37  POSITION_Defender - Left-Back          2384 non-null   int64  
 38  POSITION_Defender - Right-Back         2384 non-null   int64  
 39  POSITION_midfield - Attacking Midfield  2384 non-null   int64  
 40  POSITION_midfield - Central Midfield   2384 non-null   int64  
 41  POSITION_midfield - Defensive Midfield  2384 non-null   int64  
 42  POSITION_midfield - Left Midfield      2384 non-null   int64  
 43  POSITION_midfield - Right Midfield     2384 non-null   int64  
 44  MACRO_POSITION_Attack                  2384 non-null   int64  
 45  MACRO_POSITION_Defense                 2384 non-null   int64  
 46  MACRO_POSITION_Midfield                2384 non-null   int64  
 47  FOOT_both                              2384 non-null   int64  
 48  FOOT_left                              2384 non-null   int64  
 49  FOOT_right                             2384 non-null   int64  
 50  PLAYER_AGENT_False                     2384 non-null   int64  
 51  PLAYER_AGENT_True                      2384 non-null   int64  
 52  OUTFITTER_False                        2384 non-null   int64  
 53  OUTFITTER_True                         2384 non-null   int64  
```

```
dtypes: float64(25), int64(29)
memory usage: 1.0 MB
```

In [ ]:
```
y.tail(5)
```

Out[ ]:

	PLAYER_VALUE
2379	15.201805
2380	14.845130
2381	13.910821
2382	13.815511
2383	13.304685

In [ ]:
```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2384 entries, 0 to 2383
Data columns (total 1 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PLAYER_VALUE  2384 non-null   float64
dtypes: float64(1)
memory usage: 37.2 KB
```

Split data into training and testing

In [ ]:
```
# Split the data into training and testing sets using 40% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=
```

Define evaluate model function

In [ ]:
```
# Function to evaluate and print model performance metrics
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    print(f"RMSE: {rmse:.2f}, R^2: {r2:.2f}")
```

Modelling using train - test split

In [ ]:
```
# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

print("Linear Regression:")
evaluate_model(y_test, y_pred_lr)

# Decision Trees
dt = DecisionTreeRegressor(random_state=1)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

```
print("Decision Trees:")
evaluate_model(y_test, y_pred_dt)

# Lasso Regression
lasso = Lasso(random_state=1)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)

print("Lasso Regression:")
evaluate_model(y_test, y_pred_lasso)

# Decision Tree Regression
dt_regression = DecisionTreeRegressor(random_state=1)
dt_regression.fit(X_train, y_train)
y_pred_dt_regression = dt_regression.predict(X_test)

print("Decision Tree Regression:")
evaluate_model(y_test, y_pred_dt_regression)
```

```
Linear Regression:
RMSE: 0.83, R^2: 0.64
Decision Trees:
RMSE: 1.05, R^2: 0.42
Lasso Regression:
RMSE: 1.37, R^2: 0.00
Decision Tree Regression:
RMSE: 1.05, R^2: 0.42
```

Modelling using k-fold technique

In [ ]:

```
# Cross-validation using k-fold technique
kf = KFold(n_splits=5, random_state=1, shuffle=True)

# Linear Regression
lr_scores = cross_val_score(lr, X, y.values.ravel(), cv=kf, scoring='neg_mean_squared_
lr_rmse_cv = np.sqrt(-lr_scores.mean())
print("Linear Regression (k-fold CV):")
print(f"RMSE: {lr_rmse_cv:.2f}")

# Decision Trees
dt_scores = cross_val_score(dt, X, y.values.ravel(), cv=kf, scoring='neg_mean_squared_
dt_rmse_cv = np.sqrt(-dt_scores.mean())
print("Decision Trees (k-fold CV):")
print(f"RMSE: {dt_rmse_cv:.2f}")

# Lasso Regression
lasso_scores = cross_val_score(lasso, X, y.values.ravel(), cv=kf, scoring='neg_mean_sc
lasso_rmse_cv = np.sqrt(-lasso_scores.mean())
print("Lasso Regression (k-fold CV):")
print(f"RMSE: {lasso_rmse_cv:.2f}")

# Decision Tree Regression
dt_regression_scores = cross_val_score(dt_regression, X, y.values.ravel(), cv=kf, scor
dt_regression_rmse_cv = np.sqrt(-dt_regression_scores.mean())
print("Decision Tree Regression (k-fold CV):")
print(f"RMSE: {dt_regression_rmse_cv:.2f}")
```

```
Linear Regression (k-fold CV):
RMSE: 0.84
Decision Trees (k-fold CV):
RMSE: 1.04
Lasso Regression (k-fold CV):
RMSE: 1.40
Decision Tree Regression (k-fold CV):
RMSE: 1.04
```

Optimizing the Linear Regression Model

Bagging (Bootstrap Aggregating)

In [ ]:
```
from sklearn.ensemble import BaggingRegressor

# Create a bagging regressor using Linear Regression as the base estimator
bagging_lr = BaggingRegressor(base_estimator=lr, n_estimators=10, random_state=1)

# Perform cross-validation and calculate RMSE
bagging_lr_scores = cross_val_score(bagging_lr, X, y.values.ravel(), cv=kf, scoring='r
bagging_lr_rmse_cv = np.sqrt(-bagging_lr_scores.mean())

print("Bagging (Linear Regression) (k-fold CV):")
print(f"RMSE: {bagging_lr_rmse_cv:.2f}")
```

```
Bagging (Linear Regression) (k-fold CV):
RMSE: 0.85
```

Boosting

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Create a Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_s

# Perform cross-validation and calculate RMSE
gb_scores = cross_val_score(gb_regressor, X, y.values.ravel(), cv=kf, scoring='neg_mea
gb_rmse_cv = np.sqrt(-gb_scores.mean())

print("Gradient Boosting (k-fold CV):")
print(f"RMSE: {gb_rmse_cv:.2f}")
```

```
Gradient Boosting (k-fold CV):
RMSE: 0.68
```

Hyperparameter Optimization

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

# Define hyperparameters to search over
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3],
    # Add other hyperparameters specific to the chosen model
```

```
}

# Create a GridSearchCV object
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=1), param_grid, cv=k

# Perform the grid search
grid_search.fit(X, y.values.ravel())

# Print the best hyperparameters, RMSE, and R-squared
best_params = grid_search.best_params_
best_rmse = np.sqrt(-grid_search.best_score_)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X)
best_r2 = r2_score(y, y_pred)

print("Best Hyperparameters:", best_params)
print("Best RMSE:", best_rmse)
print("Best R-squared:", best_r2)
```

```
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}
Best RMSE: 0.6527793204881664
Best R-squared: 0.9304604540636039
```

In [ ]:

```
# Use the best hyperparameters from the grid search
best_params = {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200}

# Create the Gradient Boosting Regressor with the best hyperparameters
best_gb_regressor = GradientBoostingRegressor(**best_params, random_state=1)

# Train the model on the entire dataset
best_gb_regressor.fit(X, y.values.ravel())

# Get feature importances
feature_importances = best_gb_regressor.feature_importances_

# Create a list of (feature_name, importance) tuples
feature_importance_list = [(feature_name, importance) for feature_name, importance in

# Sort the list by importance (descending)
feature_importance_list = sorted(feature_importance_list, key=lambda x: x[1], reverse=

# Print the top 10 most important features
print("Top 10 Most Important Features:")
for feature_name, importance in feature_importance_list[:10]:
    print(f"{feature_name}: {importance:.4f}")
```

```
Top 10 Most Important Features:
TOTDIST_CARRIES: 0.1874
AGE: 0.1426
ATT PEN_TOUCHES: 0.1323
PLUS_PER__MINUS__TEAM.SUCCESS: 0.1318
LEAGUE_COUNTRY_England: 0.0964
MIN_PLAYING: 0.0590
ATT 3RD_TOUCHES: 0.0442
CLUB_NAME: 0.0426
TOTDIST_TOTAL: 0.0317
LEAGUE_COUNTRY_France: 0.0186
```

In [ ]:

```
from sklearn.model_selection import cross_val_score
```

```
# Perform cross-validation using k-fold technique
kf = KFold(n_splits=5, random_state=1, shuffle=True)

# Calculate R-squared for the entire dataset
r2_scores = cross_val_score(best_gb_regressor, X, y.values.ravel(), cv=kf, scoring='r2
average_r2 = r2_scores.mean()

print("Average R-squared on Validation Sets:", average_r2)

# Calculate R-squared for the training set
best_gb_regressor.fit(X, y.values.ravel())
y_pred_train = best_gb_regressor.predict(X)
r2_train = r2_score(y, y_pred_train)

print("R-squared on Training Set:", r2_train)

# Compare R-squared values to detect overfitting
if r2_train > average_r2:
    print("The model may be overfitting.")
else:
    print("The model appears to be performing reasonably.")
```

```
Average R-squared on Validation Sets: 0.782943751767603
R-squared on Training Set: 0.9304604540636039
The model may be overfitting.
```

Modelling based on the player's position

In [ ]:

```
# Separate the data based on 'MACRO_POSITION' columns
attack_indices = X['MACRO_POSITION_Attack'] == 1
defense_indices = X['MACRO_POSITION_Defense'] == 1
midfield_indices = X['MACRO_POSITION_Midfield'] == 1
```

Gradient Boosting

In [ ]:

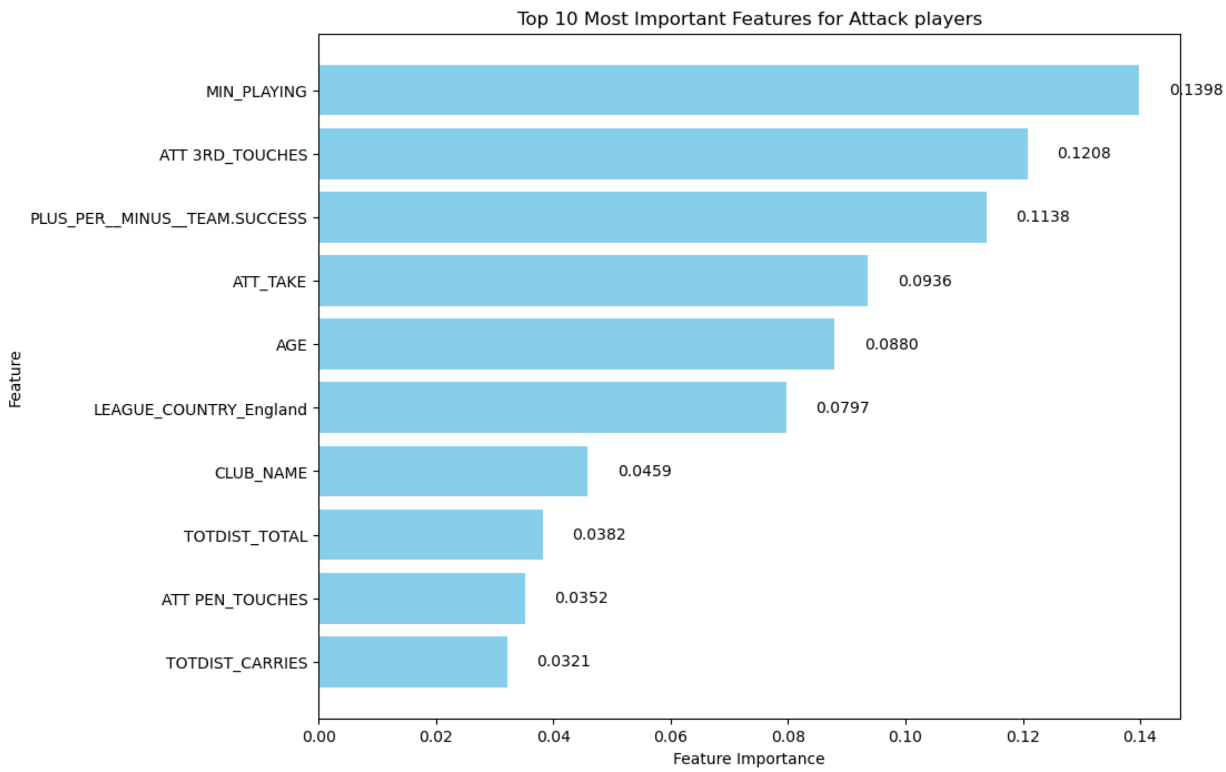
```
import joblib
import statsmodels.api as sm

# Initialize lists to store RMSE values for each model
rmse_attack = []
rmse_defense = []
rmse_midfield = []

# Initialize lists to store R-squared values for each model
r2_attack = []
r2_defense = []
r2_midfield = []

# Initialize lists to store feature importances for each model
feature_importance_attack = []
feature_importance_defense = []
feature_importance_midfield = []

# Perform hyperparameter optimization and train models for each subset
for indices, position in zip([attack_indices, defense_indices, midfield_indices], ['At
    X_subset = X[indices]
    y_subset = y[indices]
```

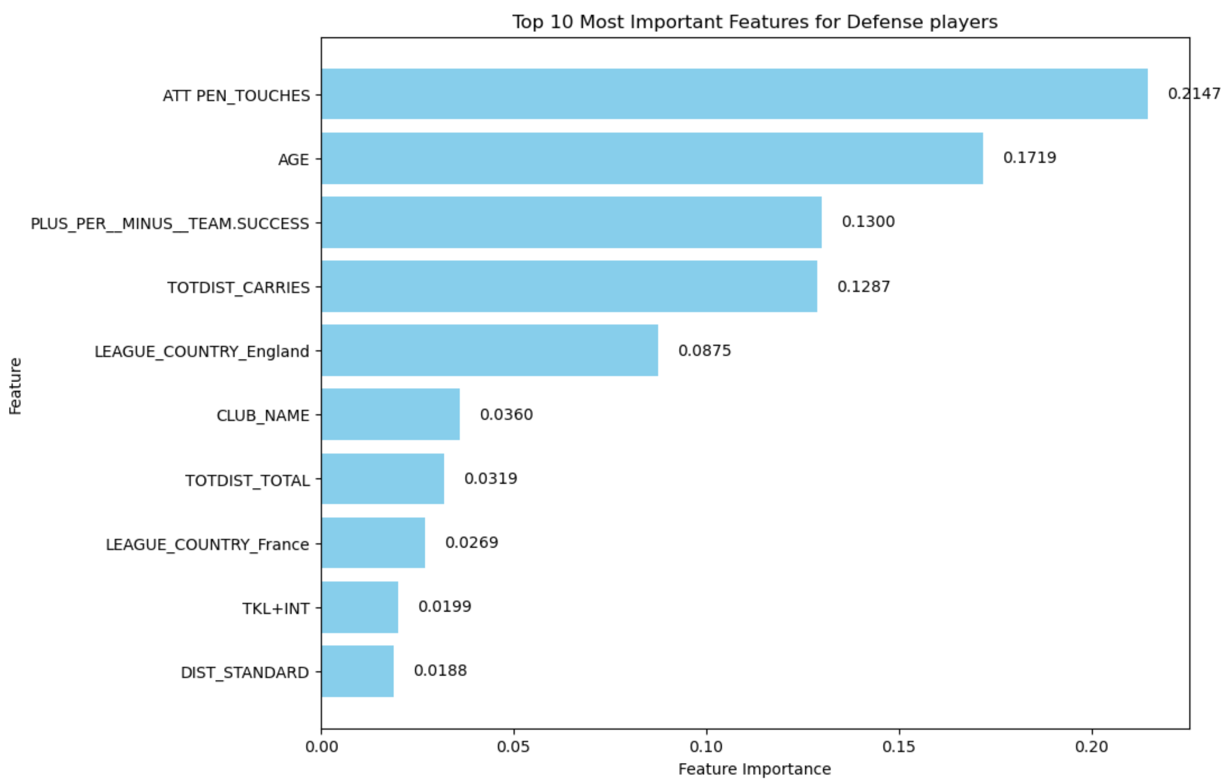
```
    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=

    # Create a Gradient Boosting Regressor
    gb_regressor = GradientBoostingRegressor(n_estimators=200, learning_rate=0.2, max_

    # Fit the model
    gb_regressor.fit(X_train, y_train.values.ravel())

    # Predict on the test set
    y_pred = gb_regressor.predict(X_test)

    # Calculate R-squared for the current subset
    r2 = r2_score(y_test, y_pred)

    if position == 'Attack':
        r2_attack.append(r2)
        feature_importance_attack.append(gb_regressor.feature_importances_)
    elif position == 'Defense':
        r2_defense.append(r2)
        feature_importance_defense.append(gb_regressor.feature_importances_)
    elif position == 'Midfield':
        r2_midfield.append(r2)
        feature_importance_midfield.append(gb_regressor.feature_importances_)

    # Calculate RMSE for the current subset
    rmse = mean_squared_error(y_test, y_pred, squared=False)

    if position == 'Attack':
        rmse_attack.append(rmse)
    elif position == 'Defense':
        rmse_defense.append(rmse)
    elif position == 'Midfield':
        rmse_midfield.append(rmse)

    # Save the model
    model_filename = f'models/model_{position}.joblib'
    joblib.dump(gb_regressor, model_filename)
```

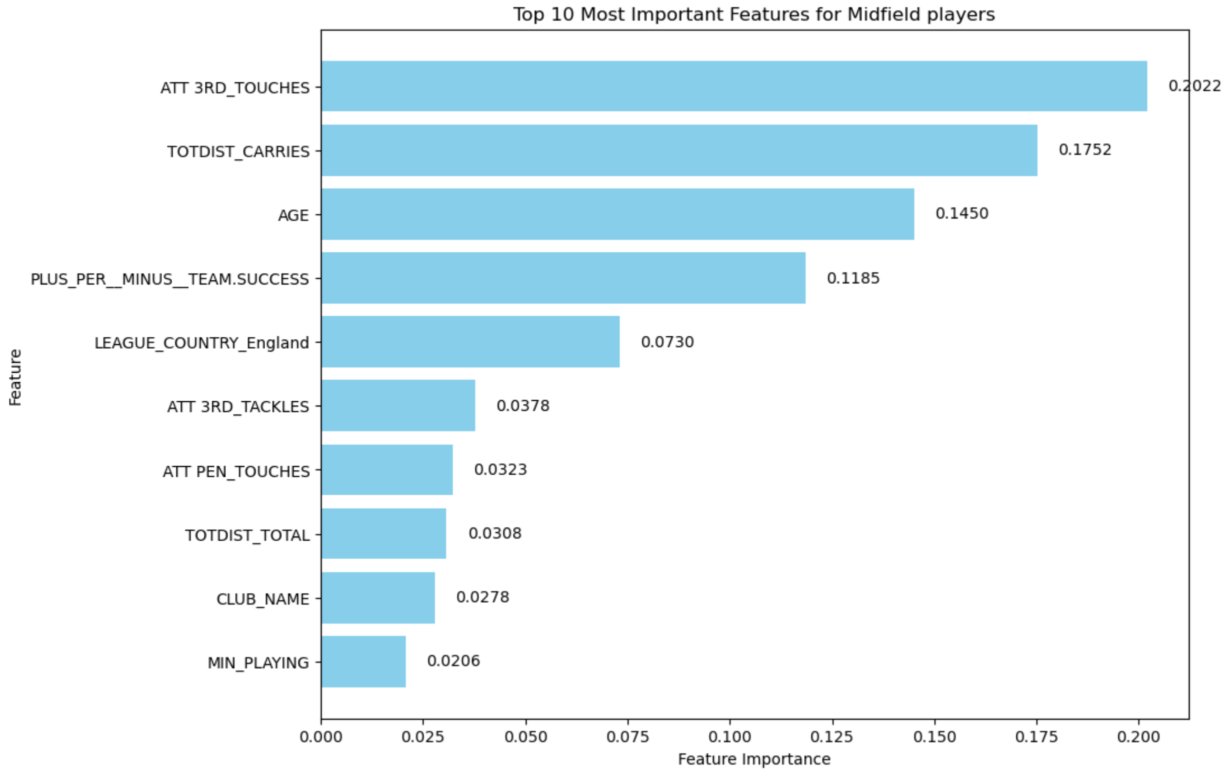
In [ ]:

```
import matplotlib.pyplot as plt

# Show the top 5 most important features for each model
def plot_feature_importance(position, feature_importance):
    top_features = X.columns[np.argsort(feature_importance)[::-1]][:10]
    top_importance = feature_importance[np.argsort(feature_importance)[::-1]][:10]

    plt.figure(figsize=(10, 8))
    plt.barh(top_features, top_importance, color='skyblue')
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature')
    plt.title(f'Top 10 Most Important Features for {position} players')
    plt.gca().invert_yaxis()

    for i, (feature, importance) in enumerate(zip(top_features, top_importance)):
        plt.text(importance + 0.005, i, f'{importance:.4f}', va='center', color='black

    plt.show()
```

```
plot_feature_importance('Attack', np.mean(feature_importance_attack, axis=0))
plot_feature_importance('Defense', np.mean(feature_importance_defense, axis=0))
plot_feature_importance('Midfield', np.mean(feature_importance_midfield, axis=0))
```







Overfitting Analysis

In [ ]:
```
from sklearn.metrics import mean_squared_error

# Initialize lists to store RMSE values for each model
rmse_attack_train = []
rmse_attack_test = []

rmse_defense_train = []
rmse_defense_test = []

rmse_midfield_train = []
rmse_midfield_test = []

# Perform hyperparameter optimization and train models for each subset
for indices, position in zip([attack_indices, defense_indices, midfield_indices], ['At
    X_subset = X[indices]
    y_subset = y[indices]

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=

    # Create a Gradient Boosting Regressor
    gb_regressor = GradientBoostingRegressor(n_estimators=200, learning_rate=0.2, max_

    # Fit the model
    gb_regressor.fit(X_train, y_train.values.ravel())

    # Predict on training data
    y_train_pred = gb_regressor.predict(X_train)

    # Predict on test data
    y_test_pred = gb_regressor.predict(X_test)
```

```
    # Calculate RMSE for training and test data
    rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

    if position == 'Attack':
        rmse_attack_train.append(rmse_train)
        rmse_attack_test.append(rmse_test)
    elif position == 'Defense':
        rmse_defense_train.append(rmse_train)
        rmse_defense_test.append(rmse_test)
    elif position == 'Midfield':
        rmse_midfield_train.append(rmse_train)
        rmse_midfield_test.append(rmse_test)

# Print RMSE values for each model
print("Overfitting Analysis - Attack:")
print(f"  RMSE (Train): {np.mean(rmse_attack_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_attack_test):.4f}")
print()

print("Overfitting Analysis - Defense:")
print(f"  RMSE (Train): {np.mean(rmse_defense_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_defense_test):.4f}")
print()

print("Overfitting Analysis - Midfield:")
print(f"  RMSE (Train): {np.mean(rmse_midfield_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_midfield_test):.4f}")
```

```
Overfitting Analysis - Attack:
  RMSE (Train): 0.0762
  RMSE (Test): 0.7927

Overfitting Analysis - Defense:
  RMSE (Train): 0.0965
  RMSE (Test): 0.6942

Overfitting Analysis - Midfield:
  RMSE (Train): 0.0981
  RMSE (Test): 0.7057
```

Ridge Regressor

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge

# Initialize lists to store RMSE values for each model
rmse_attack_train = []
rmse_attack_test = []
r2_attack_scores = []
best_params_attack = []

rmse_defense_train = []
rmse_defense_test = []
r2_defense_scores = []
best_params_defense = []

rmse_midfield_train = []
```

```
rmse_midfield_test = []
r2_midfield_scores = []
best_params_midfield = []

# Initialize lists to store feature importances for each model
feature_importance_attack = []
feature_importance_defense = []
feature_importance_midfield = []

# Parameters for grid search
param_grid = {'alpha': [0.01, 0.1, 1, 10]}

# Perform hyperparameter optimization and train models for each subset
for indices, position in zip([attack_indices, defense_indices, midfield_indices], ['At
    X_subset = X[indices]
    y_subset = y[indices]

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=

    # Create Ridge regressor
    ridge = Ridge()

    # Perform GridSearchCV
    grid_search = GridSearchCV(ridge, param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    # Get best parameters and best estimator
    best_params = grid_search.best_params_
    best_estimator = grid_search.best_estimator_

    # Predict on training data
    y_train_pred = best_estimator.predict(X_train)

    # Predict on test data
    y_test_pred = best_estimator.predict(X_test)

    # Calculate RMSE for training and test data
    rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

    # Calculate R-squared score for test data
    r2_score_test = r2_score(y_test, y_test_pred)

    if position == 'Attack':
        rmse_attack_train.append(rmse_train)
        rmse_attack_test.append(rmse_test)
        r2_attack_scores.append(r2_score_test)
        feature_importance_attack = best_estimator.coef_
        best_params_attack.append(best_params)
    elif position == 'Defense':
        rmse_defense_train.append(rmse_train)
        rmse_defense_test.append(rmse_test)
        r2_defense_scores.append(r2_score_test)
        feature_importance_defense = best_estimator.coef_
        best_params_defense.append(best_params)
    elif position == 'Midfield':
        rmse_midfield_train.append(rmse_train)
        rmse_midfield_test.append(rmse_test)
        r2_midfield_scores.append(r2_score_test)
```

```python
        feature_importance_midfield = best_estimator.coef_
        best_params_midfield.append(best_params)

# Print overfitting analysis and R-squared scores for each model
print("Overfitting Analysis with Regularization - Attack:")
print(f"  RMSE (Train): {np.mean(rmse_attack_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_attack_test):.4f}")
print(f"  R-squared (Test): {np.mean(r2_attack_scores):.4f}")
print(f"  Best Parameters: {best_params_attack}")

print("\nOverfitting Analysis with Regularization - Defense:")
print(f"  RMSE (Train): {np.mean(rmse_defense_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_defense_test):.4f}")
print(f"  R-squared (Test): {np.mean(r2_defense_scores):.4f}")
print(f"  Best Parameters: {best_params_defense}")

print("\nOverfitting Analysis with Regularization - Midfield:")
print(f"  RMSE (Train): {np.mean(rmse_midfield_train):.4f}")
print(f"  RMSE (Test): {np.mean(rmse_midfield_test):.4f}")
print(f"  R-squared (Test): {np.mean(r2_midfield_scores):.4f}")
print(f"  Best Parameters: {best_params_midfield}")
```

```
Overfitting Analysis with Regularization - Attack:
  RMSE (Train): 0.8388
  RMSE (Test): 0.8805
  R-squared (Test): 0.6567
  Best Parameters: [{'alpha': 1}]

Overfitting Analysis with Regularization - Defense:
  RMSE (Train): 0.7605
  RMSE (Test): 0.7887
  R-squared (Test): 0.6419
  Best Parameters: [{'alpha': 1}]

Overfitting Analysis with Regularization - Midfield:
  RMSE (Train): 0.7976
  RMSE (Test): 0.8522
  R-squared (Test): 0.6204
  Best Parameters: [{'alpha': 1}]
```