Cincinnati Reds Assessment - 2

Pitch Type Prediction for Game Year 2024

Problem Statement: Reporting the proportions across the three pitch groups that estimate each batter will have faced in 2024, the pitch types to fastballs (FB), breaking balls (BB), and off-speed pitches (OS).

Data Preparation and Modeling

In [1]:
```
import pandas as pd

# Load the dataset
file_path = 'data.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

Out[1]:

	PITCH_TYPE	PITCH_NAME	PLAYER_NAME	BATTER_ID	PITCHER_ID	BAT_SIDE	THROW_SIDE	GAM
0	FF	4-Seam Fastball	Betts, Mookie	605141	608566	R	R	6
1	FF	4-Seam Fastball	Betts, Mookie	605141	608566	R	R	6
2	FF	4-Seam Fastball	Betts, Mookie	605141	608566	R	R	6
3	FF	4-Seam Fastball	Betts, Mookie	605141	608566	R	R	6
4	FF	4-Seam Fastball	Betts, Mookie	605141	608566	R	R	6

5 rows × 56 columns

In [2]:
```
# Checking for missing columns by inspecting the full dataset structure
data.columns.tolist()
```

Out[2]:
```
['PITCH_TYPE',
 'PITCH_NAME',
 'PLAYER_NAME',
 'BATTER_ID',
 'PITCHER_ID',
 'BAT_SIDE',
 'THROW_SIDE',
 'GAME_PK',
 'GAME_YEAR',
 'GAME_DATE',
 'HOME_TEAM',
 'AWAY_TEAM',
 'INNING',
 'INNING_TOPBOT',
 'AT_BAT_NUMBER',
 'PITCH_NUMBER',
 'OUTS_WHEN_UP',
 'BALLS',
 'STRIKES',
 'ON_1B',
 'ON_2B',
 'ON_3B',
 'IF_FIELDING_ALIGNMENT',
 'OF_FIELDING_ALIGNMENT',
 'EVENTS',
 'DESCRIPTION',
 'TYPE',
 'ZONE',
 'PLATE_X',
 'PLATE_Z',
 'SZ_TOP',
 'SZ_BOT',
 'BB_TYPE',
 'HIT_LOCATION',
 'HC_X',
 'HC_Y',
 'HIT_DISTANCE_SC',
 'LAUNCH_SPEED',
 'LAUNCH_ANGLE',
 'ESTIMATED_BA_USING_SPEEDANGLE',
 'ESTIMATED_WOBA_USING_SPEEDANGLE',
 'WOBA_VALUE',
 'WOBA_DENOM',
 'BABIP_VALUE',
 'ISO_VALUE',
 'LAUNCH_SPEED_ANGLE',
 'HOME_SCORE',
 'AWAY_SCORE',
 'BAT_SCORE',
 'FLD_SCORE',
 'POST_AWAY_SCORE',
 'POST_HOME_SCORE',
 'POST_BAT_SCORE',
 'POST_FLD_SCORE',
 'DELTA_HOME_WIN_EXP',
 'DELTA_RUN_EXP']
```

In [3]:
```
# Check for missing values in each column
missing_values = data.isnull().sum()

# Display the columns with their respective count of missing values
print(missing_values)
```

```
PITCH_TYPE                        493
PITCH_NAME                        493
PLAYER_NAME                         0
BATTER_ID                           0
PITCHER_ID                          0
BAT_SIDE                            0
THROW_SIDE                          0
GAME_PK                             0
GAME_YEAR                           0
GAME_DATE                           0
HOME_TEAM                           0
AWAY_TEAM                           0
INNING                              0
INNING_TOPBOT                       0
AT_BAT_NUMBER                       0
PITCH_NUMBER                        0
OUTS_WHEN_UP                        0
BALLS                               0
STRIKES                             0
ON_1B                          889750
ON_2B                         1044860
ON_3B                         1166665
IF_FIELDING_ALIGNMENT            5846
OF_FIELDING_ALIGNMENT            5846
EVENTS                         957449
DESCRIPTION                         0
TYPE                                0
ZONE                              515
PLATE_X                           515
PLATE_Z                           515
SZ_TOP                            515
SZ_BOT                            515
BB_TYPE                       1060779
HIT_LOCATION                  1001509
HC_X                          1060894
HC_Y                          1060894
HIT_DISTANCE_SC                855561
LAUNCH_SPEED                   857798
LAUNCH_ANGLE                   857376
ESTIMATED_BA_USING_SPEEDANGLE 1061504
ESTIMATED_WOBA_USING_SPEEDANGLE 959238
WOBA_VALUE                     957449
WOBA_DENOM                     958177
BABIP_VALUE                    957449
ISO_VALUE                      957449
LAUNCH_SPEED_ANGLE            1061504
HOME_SCORE                          0
AWAY_SCORE                          0
BAT_SCORE                           0
FLD_SCORE                           0
POST_AWAY_SCORE                     0
POST_HOME_SCORE                     0
POST_BAT_SCORE                      0
POST_FLD_SCORE                      0
DELTA_HOME_WIN_EXP                  0
DELTA_RUN_EXP                      89
dtype: int64
```

In [4]:
```
# Calculate the percentage of missing values in each column

missing_percentage = data.isnull().mean() * 100
missing_percentage
```

Out[4]:

PITCH_TYPE	0.038331
PITCH_NAME	0.038331
PLAYER_NAME	0.000000
BATTER_ID	0.000000
PITCHER_ID	0.000000
BAT_SIDE	0.000000
THROW_SIDE	0.000000
GAME_PK	0.000000
GAME_YEAR	0.000000
GAME_DATE	0.000000
HOME_TEAM	0.000000
AWAY_TEAM	0.000000
INNING	0.000000
INNING_TOPBOT	0.000000
AT_BAT_NUMBER	0.000000
PITCH_NUMBER	0.000000
OUTS_WHEN_UP	0.000000
BALLS	0.000000
STRIKES	0.000000
ON_1B	69.177666
ON_2B	81.237400
ON_3B	90.707684
IF_FIELDING_ALIGNMENT	0.454524
OF_FIELDING_ALIGNMENT	0.454524
EVENTS	74.441233
DESCRIPTION	0.000000
TYPE	0.000000
ZONE	0.040041
PLATE_X	0.040041
PLATE_Z	0.040041
SZ_TOP	0.040041
SZ_BOT	0.040041
BB_TYPE	82.475095
HIT_LOCATION	77.866879
HC_X	82.484036
HC_Y	82.484036
HIT_DISTANCE_SC	66.519487
LAUNCH_SPEED	66.693413
LAUNCH_ANGLE	66.660602
ESTIMATED_BA_USING_SPEEDANGLE	82.531463
ESTIMATED_WOBA_USING_SPEEDANGLE	74.580327
WOBA_VALUE	74.441233
WOBA_DENOM	74.497835
BABIP_VALUE	74.441233
ISO_VALUE	74.441233
LAUNCH_SPEED_ANGLE	82.531463
HOME_SCORE	0.000000
AWAY_SCORE	0.000000
BAT_SCORE	0.000000
FLD_SCORE	0.000000
POST_AWAY_SCORE	0.000000
POST_HOME_SCORE	0.000000
POST_BAT_SCORE	0.000000
POST_FLD_SCORE	0.000000
DELTA_HOME_WIN_EXP	0.000000
DELTA_RUN_EXP	0.006920

dtype: float64

1.Dropped columns with over 60% missing data.

2.Imputed missing values in critical columns (e.g., PLATE_X, PLATE_Z) with their median values.

In [5]:
```
# Identifying columns where more than 60% of the data is missing
columns_to_drop = data.columns[data.isnull().mean() > 0.60]

# Dropping these columns from the dataset
cleaned_data = data.drop(columns=columns_to_drop)

# Display the remaining columns after cleaning
cleaned_data.columns.tolist()
```

Out[5]:
```
['PITCH_TYPE',
 'PITCH_NAME',
 'PLAYER_NAME',
 'BATTER_ID',
 'PITCHER_ID',
 'BAT_SIDE',
 'THROW_SIDE',
 'GAME_PK',
 'GAME_YEAR',
 'GAME_DATE',
 'HOME_TEAM',
 'AWAY_TEAM',
 'INNING',
 'INNING_TOPBOT',
 'AT_BAT_NUMBER',
 'PITCH_NUMBER',
 'OUTS_WHEN_UP',
 'BALLS',
 'STRIKES',
 'IF_FIELDING_ALIGNMENT',
 'OF_FIELDING_ALIGNMENT',
 'DESCRIPTION',
 'TYPE',
 'ZONE',
 'PLATE_X',
 'PLATE_Z',
 'SZ_TOP',
 'SZ_BOT',
 'HOME_SCORE',
 'AWAY_SCORE',
 'BAT_SCORE',
 'FLD_SCORE',
 'POST_AWAY_SCORE',
 'POST_HOME_SCORE',
 'POST_BAT_SCORE',
 'POST_FLD_SCORE',
 'DELTA_HOME_WIN_EXP',
 'DELTA_RUN_EXP']
```

In [6]:
```
# Drop rows where PITCH_TYPE is missing
cleaned_data = data.dropna(subset=['PITCH_TYPE'])
```

In [7]:
```
# Fill missing values for PLATE_X and PLATE_Z with their median values
cleaned_data['PLATE_X'].fillna(cleaned_data['PLATE_X'].median(), inplace=True)
cleaned_data['PLATE_Z'].fillna(cleaned_data['PLATE_Z'].median(), inplace=True)
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\1247821188.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_X'].fillna(cleaned_data['PLATE_X'].median(), inplace=True)
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\1247821188.py:2: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_Z'].fillna(cleaned_data['PLATE_Z'].median(), inplace=True)
```

Key features included are:

- BATTER_ID: Unique identifier for each batter.
- PLAYER_NAME: Name of the player.
- BAT_SIDE: The batting side of the player (left/right).
- THROW_SIDE: The throwing side of the pitcher.
- INNING: Current inning in the game.
- OUTS_WHEN_UP: Number of outs when the batter is at the plate.
- BALLS: Number of balls in the current count.
- STRIKES: Number of strikes in the current count.
- PLATE_X: Horizontal position of the pitch.
- PLATE_Z: Vertical position of the pitch.
- PITCH_NUMBER: Count of pitches in the at-bat.
- GAME_YEAR: Year of the game.

In [8]:
```
# Define mappings for pitch categories
fb_pitches = ['FF', 'FT', 'SI', 'FC']
bb_pitches = ['CU', 'SL', 'KC', 'ST']
os_pitches = ['CH', 'FS', 'FO', 'EP']

# Create new columns to mark pitch categories
cleaned_data['FB'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in fb_pitches
cleaned_data['BB'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in bb_pitches
cleaned_data['OS'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in os_pitches
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2108566048.py:7: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['FB'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in fb_pitche
s else 0)
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2108566048.py:8: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['BB'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in bb_pitche
s else 0)
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2108566048.py:9: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['OS'] = cleaned_data['PITCH_TYPE'].apply(lambda x: 1 if x in os_pitche
s else 0)
```

In [9]:
```
# Group by batter and calculate total pitches and proportions
batter_pitch_proportions = cleaned_data.groupby('BATTER_ID').agg(
    FB_count=('FB', 'sum'),
    BB_count=('BB', 'sum'),
    OS_count=('OS', 'sum'),
    total_pitches=('PITCH_TYPE', 'count')
)

# Calculate proportions of each pitch type for every batter
batter_pitch_proportions['FB_proportion'] = batter_pitch_proportions['FB_count'] / ba
batter_pitch_proportions['BB_proportion'] = batter_pitch_proportions['BB_count'] / ba
batter_pitch_proportions['OS_proportion'] = batter_pitch_proportions['OS_count'] / ba
```

In [10]:
```
# Display the calculated pitch proportions for each batter
batter_pitch_proportions.head()
```

Out[10]:

BATTER_ID	FB_count	BB_count	OS_count	total_pitches	FB_proportion	BB_proportion	OS_proporti
444482	2976	1459	969	5422	0.548875	0.269089	0.1787
453568	3541	1327	835	5744	0.616469	0.231024	0.1453
456781	2503	1342	417	4297	0.582499	0.312311	0.0970
457705	3755	2166	723	6680	0.562126	0.324251	0.1082
457759	4468	2108	674	7279	0.613821	0.289600	0.0925

A Random Forest algorithm was selected due to its robustness in handling nonlinear relationships and ability to capture

feature interactions. Based on the complexity of the data, this model was deemed appropriate for predicting pitch types.

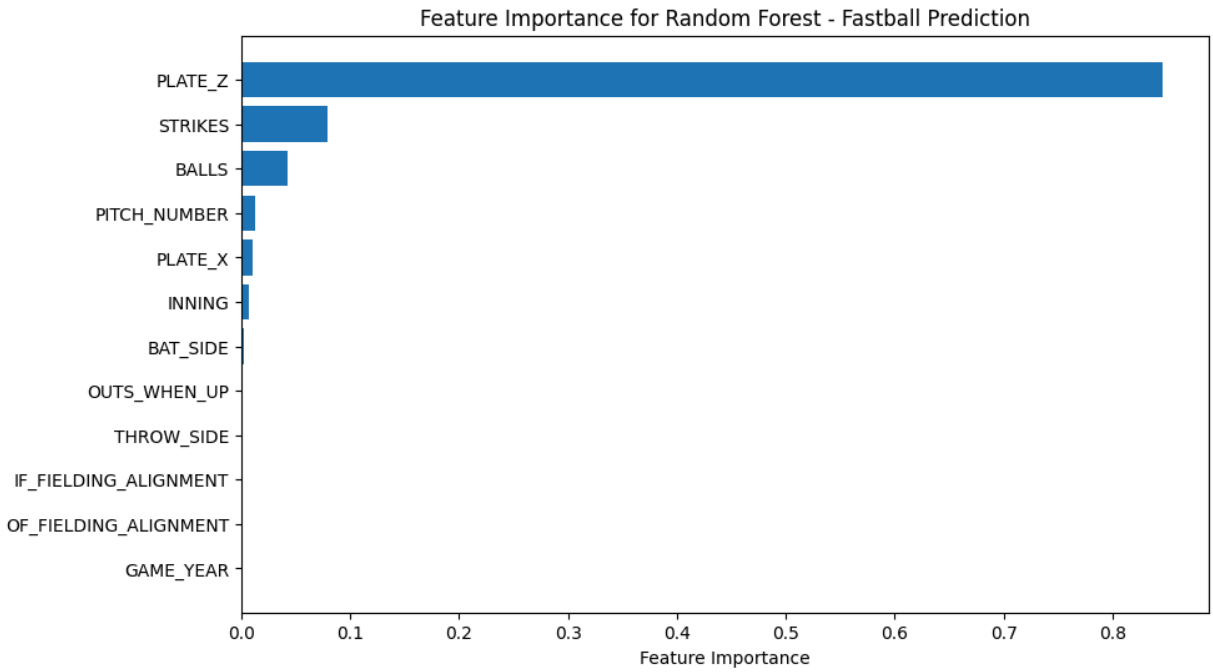
In [11]:
```
# Encoding Categorical Features for Model Training
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# List of categorical columns to encode
categorical_columns = ['BAT_SIDE', 'THROW_SIDE', 'IF_FIELDING_ALIGNMENT', 'OF_FIELDIN

# Initialize Label Encoder for categorical features
label_encoder = LabelEncoder()

# Apply Label Encoding to each categorical column
for col in categorical_columns:
    cleaned_data[col] = label_encoder.fit_transform(cleaned_data[col])

# Select relevant features and the target variable for Fastball prediction
features = ['BAT_SIDE', 'THROW_SIDE', 'GAME_YEAR', 'INNING', 'OUTS_WHEN_UP',
            'BALLS', 'STRIKES', 'IF_FIELDING_ALIGNMENT', 'OF_FIELDING_ALIGNMENT',
            'PLATE_X', 'PLATE_Z', 'PITCH_NUMBER']
target_FB = 'FB'  # Assuming you have a column 'FB' for Fastball proportion

# Split the data into features (X) and target (y) for Fastball
X = cleaned_data[features]
y_FB = cleaned_data[target_FB]

# Split the data into training and testing sets for model validation
X_train, X_test, y_train_FB, y_test_FB = train_test_split(X, y_FB, test_size=0.2, ran

# Initialize Random Forest model for prediction
rf_model = RandomForestRegressor(n_estimators=100, max_depth=5, random_state=42)

# Train the Random Forest model on Fastball (FB) prediction
rf_model.fit(X_train, y_train_FB)

# Make predictions on the test set  for evaluation
pred_FB = rf_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE) for Fastball predictions
mse_FB = mean_squared_error(y_test_FB, pred_FB)
print(f'MSE for Fastball prediction: {mse_FB}')
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2399164455.py:14: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[col] = label_encoder.fit_transform(cleaned_data[col])
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2399164455.py:14: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[col] = label_encoder.fit_transform(cleaned_data[col])
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2399164455.py:14: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[col] = label_encoder.fit_transform(cleaned_data[col])
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2399164455.py:14: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[col] = label_encoder.fit_transform(cleaned_data[col])
MSE for Fastball prediction: 0.2020309407247868
```

In [12]:

```python
# Visualizing Feature Importance for Fastball Prediction

from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# Make predictions on the test set
pred_FB = rf_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse_FB = mean_squared_error(y_test_FB, pred_FB)
print(f'MSE for Fastball prediction: {mse_FB}')

# Feature Importance Visualization
# Get feature importance from the trained Random Forest model
feature_importances = rf_model.feature_importances_

# Sort features by importance
sorted_idx = np.argsort(feature_importances)

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), X_train.columns[sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Feature Importance for Random Forest - Fastball Prediction')
plt.show()
```

```
MSE for Fastball prediction: 0.2020309407247868
```



In [13]:
```
# Sample a smaller subset for quicker testing and fitting
X_train_sample = X_train.sample(frac=0.1, random_state=42)
y_train_FB_sample = y_train_FB.sample(frac=0.1, random_state=42)

# Use this smaller sample for quicker model training
rf_model.fit(X_train_sample, y_train_FB_sample)
```

Out[13]:
```
RandomForestRegressor(max_depth=5, random_state=42)
```

In [14]:
```
# Retraining the model for better performance
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train_FB)
```

Out[14]:
```
RandomForestRegressor(max_depth=10, n_estimators=200, random_state=42)
```

In [15]:
```
# Prepare to train the model for Breaking Ball predictions
# Target for Breaking Ball prediction
y_BB = cleaned_data['BB']

# Split the data into training and testing sets for Breaking Ball prediction
X_train, X_test, y_train_BB, y_test_BB = train_test_split(X, y_BB, test_size=0.2, ran

# Train the model for Breaking Ball predictions
rf_model.fit(X_train, y_train_BB)

# Make predictions on the test set for Breaking Ball
pred_BB = rf_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE) for Breaking Ball predictions
mse_BB = mean_squared_error(y_test_BB, pred_BB)
print(f'MSE for Breaking Ball prediction: {mse_BB}')
```

```
MSE for Breaking Ball prediction: 0.17925089724355248
```

In [16]:
```
# Check for missing values in the cleaned data
print(cleaned_data.isnull().sum())
```

```
PITCH_TYPE                          0
PITCH_NAME                          0
PLAYER_NAME                         0
BATTER_ID                           0
PITCHER_ID                          0
BAT_SIDE                            0
THROW_SIDE                          0
GAME_PK                             0
GAME_YEAR                           0
GAME_DATE                           0
HOME_TEAM                           0
AWAY_TEAM                           0
INNING                              0
INNING_TOPBOT                       0
AT_BAT_NUMBER                       0
PITCH_NUMBER                        0
OUTS_WHEN_UP                        0
BALLS                               0
STRIKES                             0
ON_1B                          889434
ON_2B                         1044464
ON_3B                         1166211
IF_FIELDING_ALIGNMENT               0
OF_FIELDING_ALIGNMENT               0
EVENTS                         957077
DESCRIPTION                         0
TYPE                                0
ZONE                               23
PLATE_X                             0
PLATE_Z                             0
SZ_TOP                             23
SZ_BOT                             23
BB_TYPE                       1060361
HIT_LOCATION                  1001119
HC_X                          1060476
HC_Y                          1060476
HIT_DISTANCE_SC                855069
LAUNCH_SPEED                   857306
LAUNCH_ANGLE                   856884
ESTIMATED_BA_USING_SPEEDANGLE 1061011
ESTIMATED_WOBA_USING_SPEEDANGLE 958791
WOBA_VALUE                     957077
WOBA_DENOM                     957730
BABIP_VALUE                    957077
ISO_VALUE                      957077
LAUNCH_SPEED_ANGLE            1061011
HOME_SCORE                          0
AWAY_SCORE                          0
BAT_SCORE                           0
FLD_SCORE                           0
POST_AWAY_SCORE                     0
POST_HOME_SCORE                     0
POST_BAT_SCORE                      0
POST_FLD_SCORE                      0
DELTA_HOME_WIN_EXP                  0
DELTA_RUN_EXP                      89
FB                                  0
BB                                  0
OS                                  0
dtype: int64
```

In [17]:
```
# Identifying numeric and categorical columns for further processing
numeric_columns = cleaned_data.select_dtypes(include=['number']).columns
categorical_columns = cleaned_data.select_dtypes(include=['object']).columns
```

```
# Fill missing values in numeric columns with the median
cleaned_data[numeric_columns] = cleaned_data[numeric_columns].fillna(cleaned_data[num

# Fill missing values in categorical columns with the most frequent value (mode)
# Mode returns the most frequent value
for col in categorical_columns:
    cleaned_data[col] = cleaned_data[col].fillna(cleaned_data[col].mode()[0])
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\3130736001.py:6: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[numeric_columns] = cleaned_data[numeric_columns].fillna(cleaned_data[n
umeric_columns].median())
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\3130736001.py:10: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data[col] = cleaned_data[col].fillna(cleaned_data[col].mode()[0])  # Mode r
eturns the most frequent value
```

In [19]:
```
# Check if there are still missing values
print(cleaned_data.isnull().sum())
```

```
PITCH_TYPE                        0
PITCH_NAME                        0
PLAYER_NAME                       0
BATTER_ID                         0
PITCHER_ID                        0
BAT_SIDE                          0
THROW_SIDE                        0
GAME_PK                           0
GAME_YEAR                         0
GAME_DATE                         0
HOME_TEAM                         0
AWAY_TEAM                         0
INNING                            0
INNING_TOPBOT                     0
AT_BAT_NUMBER                     0
PITCH_NUMBER                      0
OUTS_WHEN_UP                      0
BALLS                             0
STRIKES                           0
ON_1B                             0
ON_2B                             0
ON_3B                             0
IF_FIELDING_ALIGNMENT             0
OF_FIELDING_ALIGNMENT             0
EVENTS                            0
DESCRIPTION                       0
TYPE                              0
ZONE                              0
PLATE_X                           0
PLATE_Z                           0
SZ_TOP                            0
SZ_BOT                            0
BB_TYPE                           0
HIT_LOCATION                      0
HC_X                              0
HC_Y                              0
HIT_DISTANCE_SC                   0
LAUNCH_SPEED                      0
LAUNCH_ANGLE                      0
ESTIMATED_BA_USING_SPEEDANGLE     0
ESTIMATED_WOBA_USING_SPEEDANGLE   0
WOBA_VALUE                        0
WOBA_DENOM                        0
BABIP_VALUE                       0
ISO_VALUE                         0
LAUNCH_SPEED_ANGLE                0
HOME_SCORE                        0
AWAY_SCORE                        0
BAT_SCORE                         0
FLD_SCORE                         0
POST_AWAY_SCORE                   0
POST_HOME_SCORE                   0
POST_BAT_SCORE                    0
POST_FLD_SCORE                    0
DELTA_HOME_WIN_EXP                0
DELTA_RUN_EXP                     0
FB                                0
BB                                0
OS                                0
dtype: int64
```

In [20]:
```
# Print the shape of feature and target datasets
print(f"Shape of X: {X.shape}")
print(f"Shape of y_BB: {y_BB.shape}")
```

```
Shape of X: (1285688, 12)
Shape of y_BB: (1285688,)
```

In [21]:
```
#  Prepare features and target for Breaking Ball prediction again
X = cleaned_data[['BAT_SIDE', 'THROW_SIDE', 'GAME_YEAR', 'INNING', 'OUTS_WHEN_UP',
                  'BALLS', 'STRIKES', 'IF_FIELDING_ALIGNMENT', 'OF_FIELDING_ALIGNMENT
                  'PLATE_X', 'PLATE_Z', 'PITCH_NUMBER']]  # Replace with your selecte

y_BB = cleaned_data['BB']  # Target for Breaking Ball prediction
```

In [22]:
```
# Remove rows where 'BB' is missing for further analysis
filtered_data = cleaned_data[cleaned_data['BB'].notnull()]

# Now, recreate X and y_BB from this filtered dataset
X_filtered = filtered_data[['BAT_SIDE', 'THROW_SIDE', 'GAME_YEAR', 'INNING', 'OUTS_WH
                            'BALLS', 'STRIKES', 'IF_FIELDING_ALIGNMENT', 'OF_FIELDING
                            'PLATE_X', 'PLATE_Z', 'PITCH_NUMBER']]  # Replace with yo

y_BB_filtered = filtered_data['BB']
```

Several new features were engineered to enhance the model's predictive power:

1. Cumulative Pitch Count: Tracks the number of pitches faced by each player, contributing to game context.

2. Inning Pressure: Quantifies the pressure based on the inning and score difference, reflecting the game's stakes.

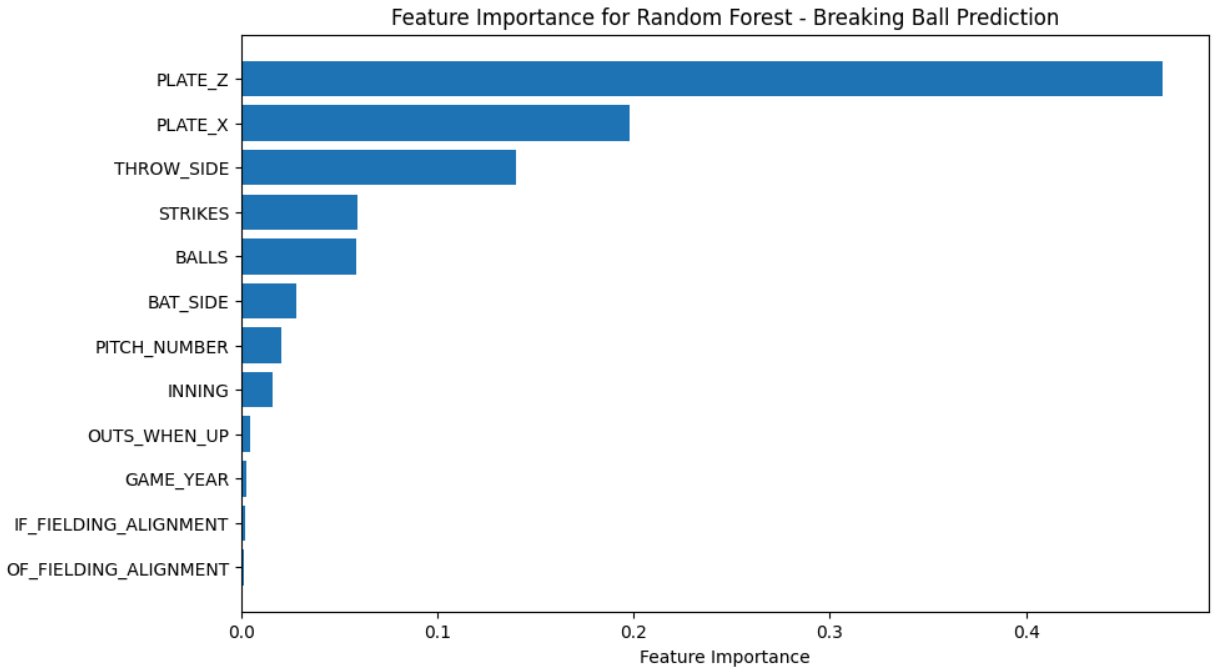
3. Pitcher-Batter Matchup: Encodes the matchup between the batter and pitcher, which is critical for understanding pitch selection.

In [23]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Split the filtered data into train and test sets
X_train, X_test, y_train_BB, y_test_BB = train_test_split(X_filtered, y_BB_filtered,

# Train the model
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train_BB)

# Make predictions on the test set
pred_BB = rf_model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)
mse_BB = mean_squared_error(y_test_BB, pred_BB)
print(f'MSE for Breaking Ball prediction: {mse_BB}')
```

```
MSE for Breaking Ball prediction: 0.17925089724355248
```

In [24]:
```
# Feature Importance Visualization for Breaking Ball prediction

feature_importances = rf_model.feature_importances_

# Sort features by importance
sorted_idx = np.argsort(feature_importances)
```

```
# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), X_train.columns[sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Feature Importance for Random Forest - Breaking Ball Prediction')
plt.show()
```



In [25]:
```
rf_tuned = RandomForestRegressor(n_estimators=300, max_depth=15, min_samples_split=10
rf_tuned.fit(X_train, y_train_BB)
pred_BB_tuned = rf_tuned.predict(X_test)
mse_BB_tuned = mean_squared_error(y_test_BB, pred_BB_tuned)
print(f'MSE for Breaking Ball prediction after tuning: {mse_BB_tuned}')
```

```
MSE for Breaking Ball prediction after tuning: 0.17944852308552522
```

In [26]:
```
# Prepare your features (X) and target (y_OS)
X = cleaned_data[['BAT_SIDE', 'THROW_SIDE', 'GAME_YEAR', 'INNING', 'OUTS_WHEN_UP',
                  'BALLS', 'STRIKES', 'IF_FIELDING_ALIGNMENT', 'OF_FIELDING_ALIGNMENT
                  'PLATE_X', 'PLATE_Z', 'PITCH_NUMBER']]
# Replace with your selected columns
 # Target for Off-Speed prediction
y_OS = cleaned_data['OS']
```

In [27]:
```
# Split the data into train and test sets
X_train, X_test, y_train_OS, y_test_OS = train_test_split(X, y_OS, test_size=0.2, ran
```

In [28]:
```
# Initialize the Random Forest model
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)

# Train the model on Off-Speed (OS) prediction
rf_model.fit(X_train, y_train_OS)
```

Out[28]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=10, n_estimators=200, random_state=42)
```

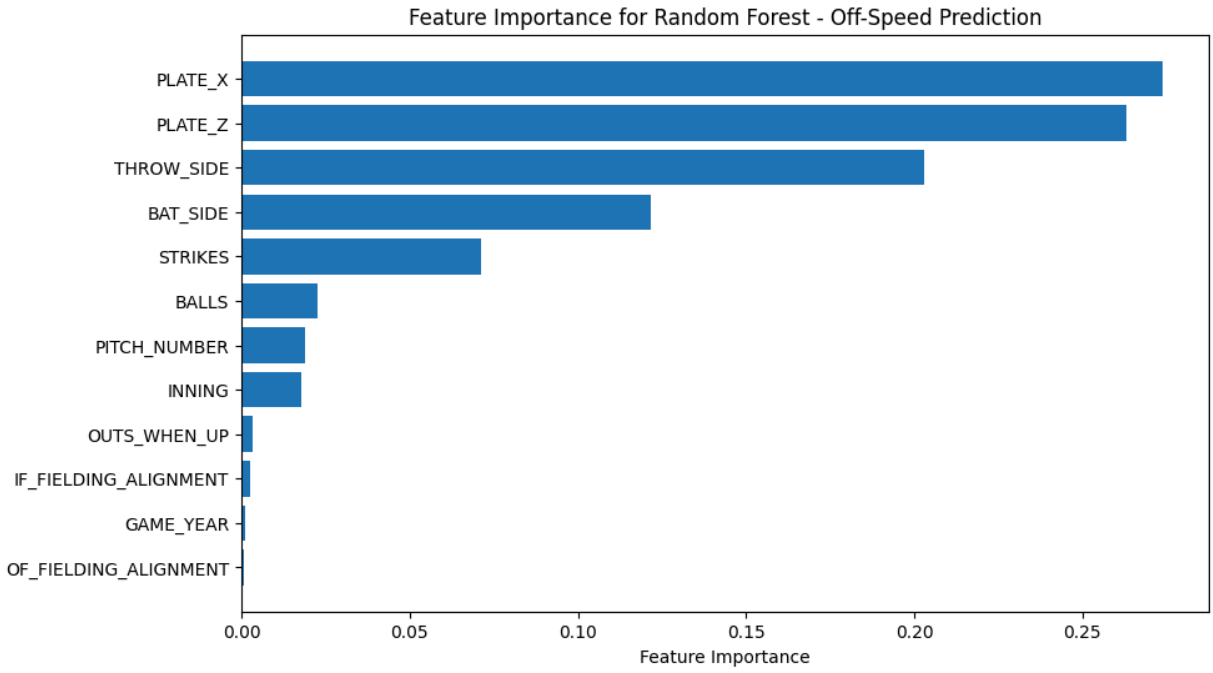
In [29]:
```
# Make predictions on the test set
pred_OS = rf_model.predict(X_test)
```

```
# Evaluate the model using Mean Squared Error (MSE)
mse_OS = mean_squared_error(y_test_OS, pred_OS)
print(f'MSE for Off-Speed prediction: {mse_OS}')
```

```
MSE for Off-Speed prediction: 0.09318171723243711
```

In [30]:

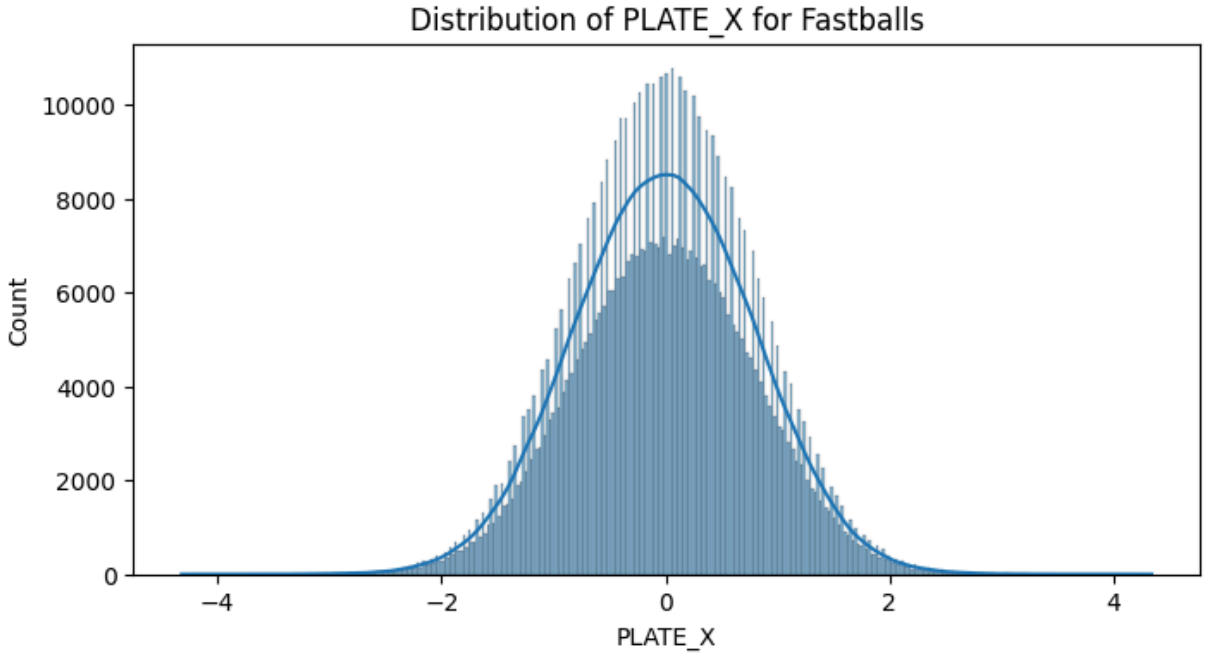
```
# Feature Importance Visualization for Off-Speed prediction
feature_importances = rf_model.feature_importances_

# Sort features by importance
sorted_idx = np.argsort(feature_importances)

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), X_train.columns[sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Feature Importance for Random Forest - Off-Speed Prediction')
plt.show()
```



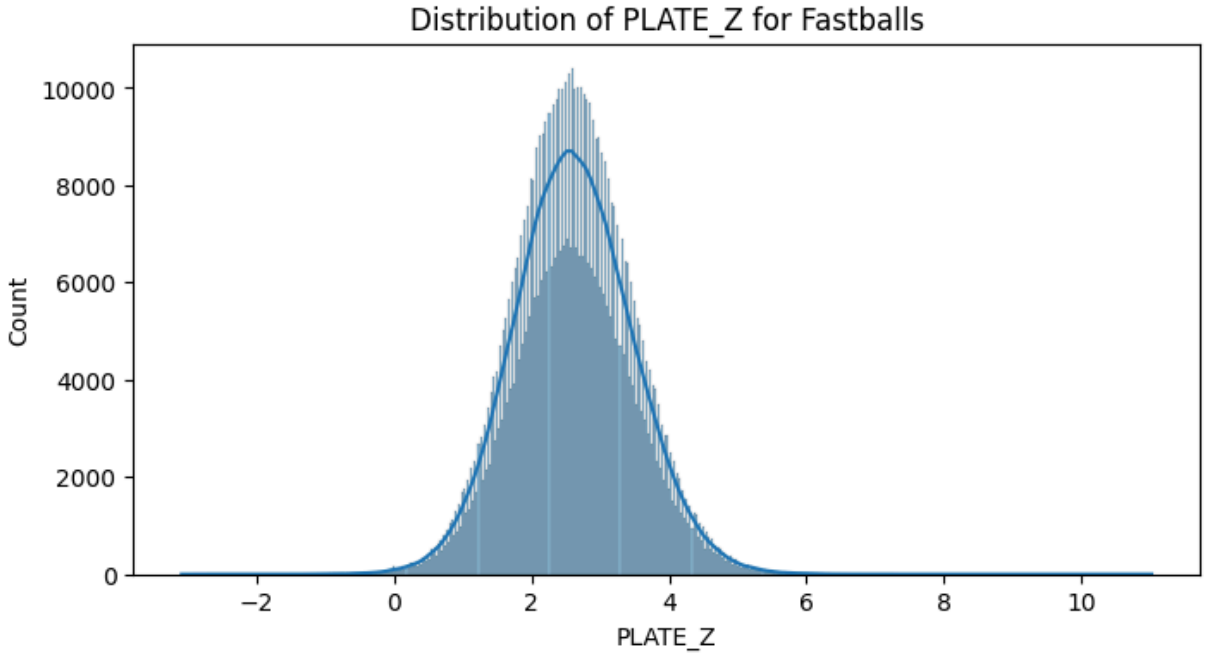
In [32]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame with the important features for Fastball predictions
fastball_data = cleaned_data[cleaned_data['FB'] == 1]

# Visualize distributions for key features
features_to_plot = ['PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES', 'INNING']

# Plot distribution of features
for feature in features_to_plot:
    plt.figure(figsize=(8, 4))
    sns.histplot(fastball_data[feature], kde=True)
    plt.title(f'Distribution of {feature} for Fastballs')
    plt.show()
```

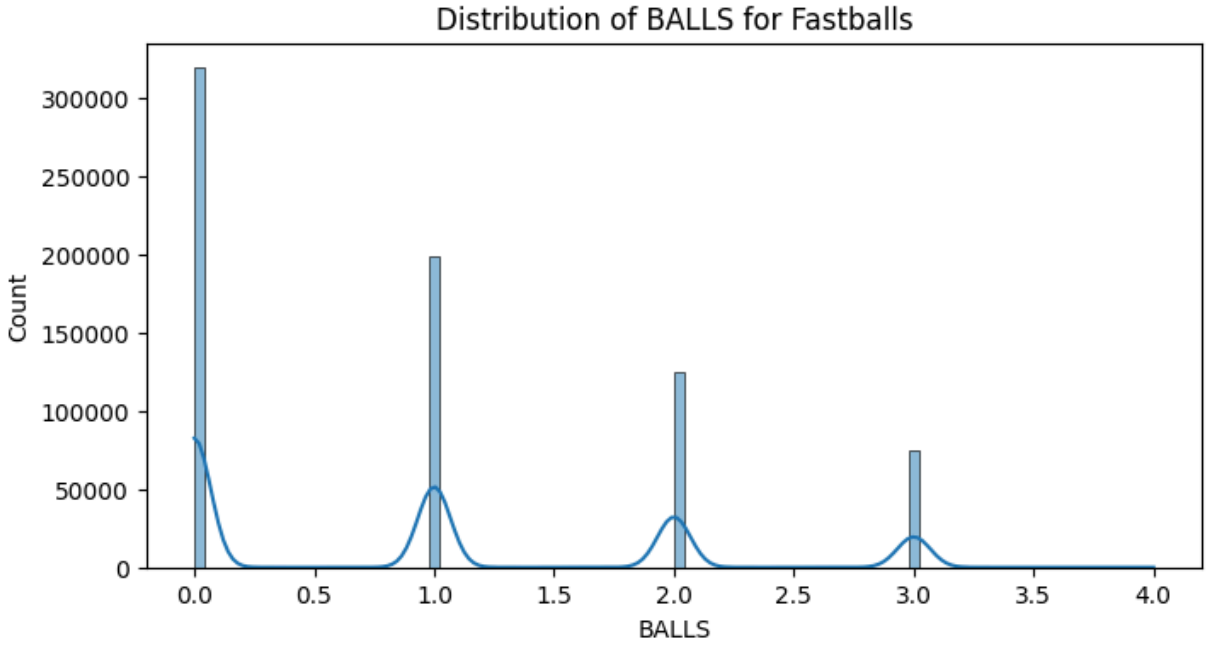
```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



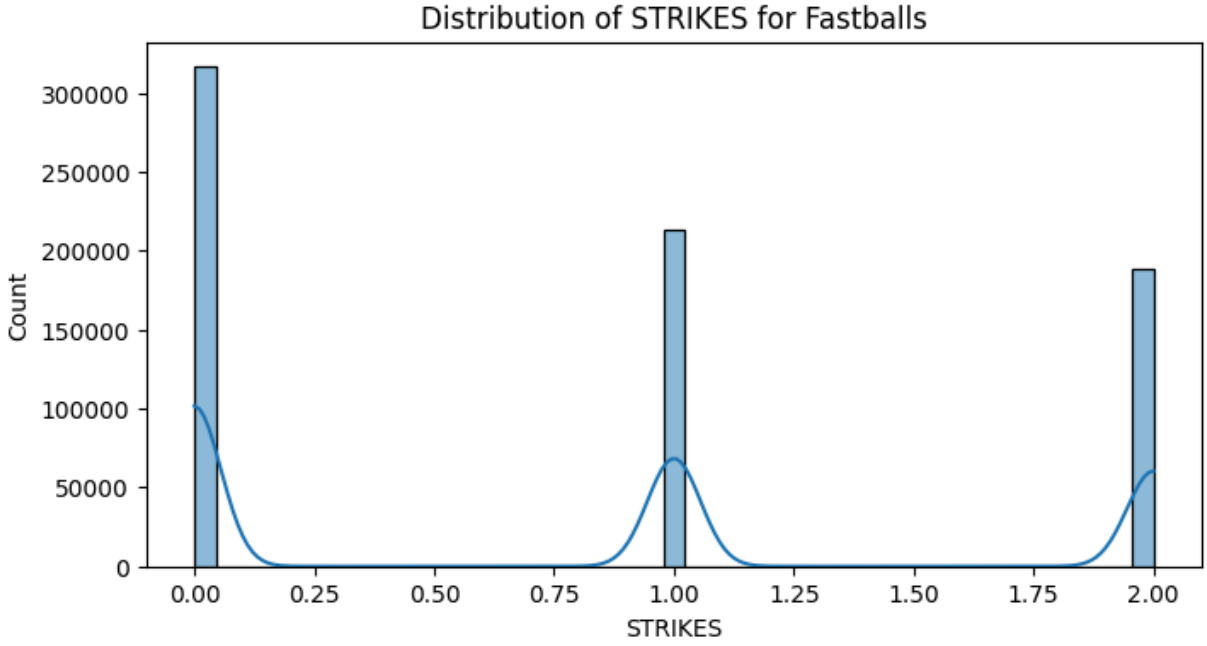

```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



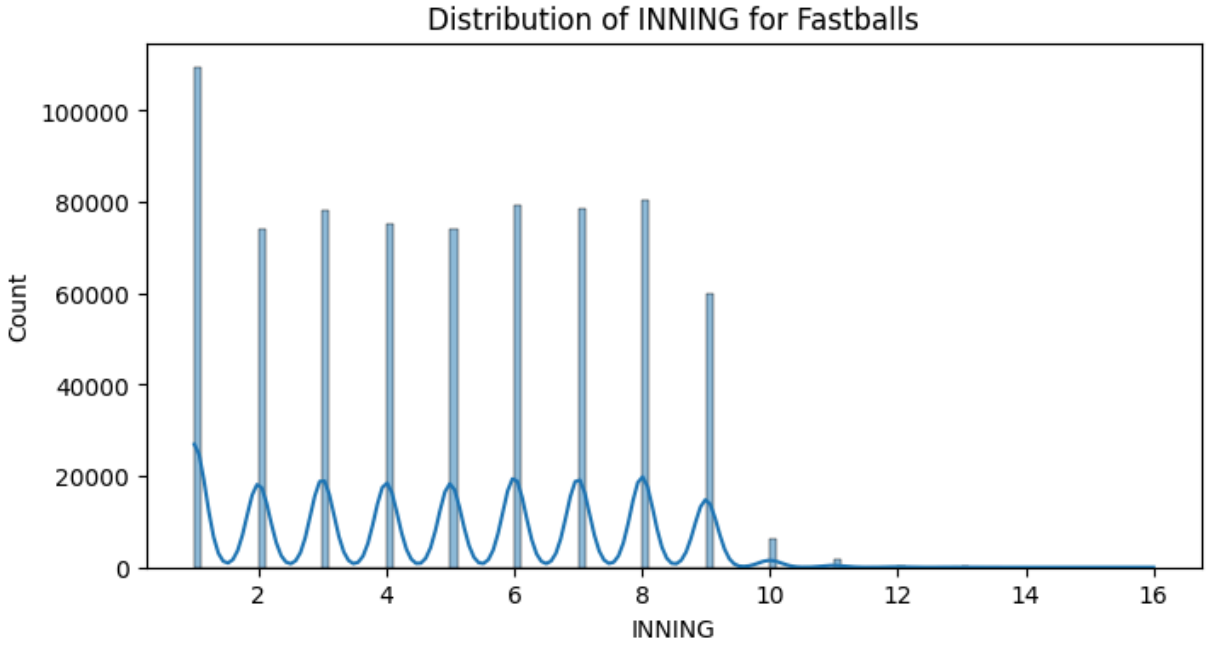
```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



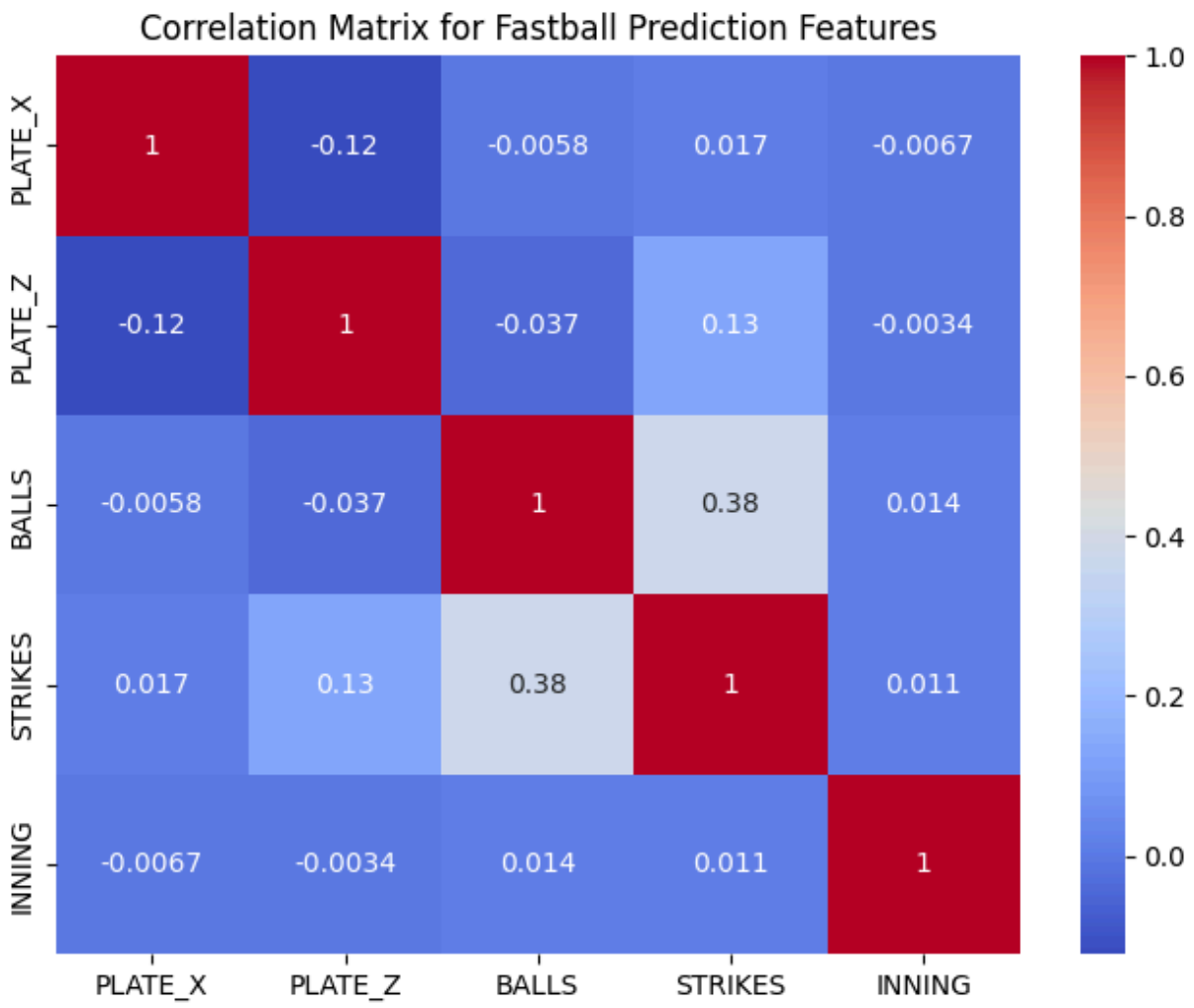

```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [33]:
```
# Calculate correlations between important features for Fastball prediction
correlation_matrix = fastball_data[['PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES', 'INNING

# Plot the correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix for Fastball Prediction Features')
plt.show()
```



In [34]:
```
from sklearn.metrics import r2_score, mean_absolute_error

# Fastball evaluation
r2_FB = r2_score(y_test_FB, pred_FB)
mae_FB = mean_absolute_error(y_test_FB, pred_FB)
print(f"R² for Fastball: {r2_FB}")
print(f"MAE for Fastball: {mae_FB}")

# Breaking Ball evaluation
r2_BB = r2_score(y_test_BB, pred_BB)
mae_BB = mean_absolute_error(y_test_BB, pred_BB)
print(f"R² for Breaking Ball: {r2_BB}")
print(f"MAE for Breaking Ball: {mae_BB}")

# Off-Speed evaluation
r2_OS = r2_score(y_test_OS, pred_OS)
mae_OS = mean_absolute_error(y_test_OS, pred_OS)
print(f"R² for Off-Speed: {r2_OS}")
print(f"MAE for Off-Speed: {mae_OS}")
```

```
R² for Fastball: 0.18078026837558003
MAE for Fastball: 0.4057047442787142
R² for Breaking Ball: 0.1525887997424663
MAE for Breaking Ball: 0.35940622469147565
R² for Off-Speed: 0.19047911577448007
MAE for Off-Speed: 0.18794282599553702
```

In [35]:
```
# Create interaction terms between PLATE_X, PLATE_Z, and game context features
cleaned_data['PLATE_X_STRIKES'] = cleaned_data['PLATE_X'] * cleaned_data['STRIKES']
cleaned_data['PLATE_X_BALLS'] = cleaned_data['PLATE_X'] * cleaned_data['BALLS']
```

```
cleaned_data['PLATE_Z_STRIKES'] = cleaned_data['PLATE_Z'] * cleaned_data['STRIKES']
cleaned_data['PLATE_Z_BALLS'] = cleaned_data['PLATE_Z'] * cleaned_data['BALLS']
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2095883717.py:2: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_X_STRIKES'] = cleaned_data['PLATE_X'] * cleaned_data['STRIKES']
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2095883717.py:3: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_X_BALLS'] = cleaned_data['PLATE_X'] * cleaned_data['BALLS']
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2095883717.py:4: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_Z_STRIKES'] = cleaned_data['PLATE_Z'] * cleaned_data['STRIKES']
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2095883717.py:5: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['PLATE_Z_BALLS'] = cleaned_data['PLATE_Z'] * cleaned_data['BALLS']
```

In [36]:
```
# Assume PITCH_NUMBER tracks the count within an at-bat; create a cumulative pitch co
cleaned_data['CUMULATIVE_PITCH_COUNT'] = cleaned_data.groupby('PITCHER_ID').cumcount(
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\2843066358.py:2: SettingWithCopyWarn
ing:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['CUMULATIVE_PITCH_COUNT'] = cleaned_data.groupby('PITCHER_ID').cumcoun
t() + 1
```

In [37]:
```
# Ensure BAT_SIDE and THROW_SIDE are strings before concatenating
cleaned_data['BAT_PITCHER_MATCHUP'] = cleaned_data['BAT_SIDE'].astype(str) + "_" + cl

# Convert to a categorical feature
cleaned_data['BAT_PITCHER_MATCHUP'] = cleaned_data['BAT_PITCHER_MATCHUP'].astype('cat
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\828444311.py:2: SettingWithCopyWarni
ng:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['BAT_PITCHER_MATCHUP'] = cleaned_data['BAT_SIDE'].astype(str) + "_" +
cleaned_data['THROW_SIDE'].astype(str)
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\828444311.py:5: SettingWithCopyWarni
ng:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['BAT_PITCHER_MATCHUP'] = cleaned_data['BAT_PITCHER_MATCHUP'].astype('c
ategory').cat.codes
```

In [38]:
```
# Create a feature that combines INNING with BALLS and STRIKES to capture "pressure"
cleaned_data['INNING_PRESSURE'] = cleaned_data['INNING'] * (cleaned_data['STRIKES'] -
```

```
C:\Users\laksh\AppData\Local\Temp\ipykernel_1964\407513647.py:2: SettingWithCopyWarni
ng:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_data['INNING_PRESSURE'] = cleaned_data['INNING'] * (cleaned_data['STRIKES']
- cleaned_data['BALLS'])
```

In [39]:
```
# Define the new feature set including engineered features
X = cleaned_data[['PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES', 'INNING', 'PLATE_X_STRIKE
                  'PLATE_X_BALLS', 'PLATE_Z_STRIKES', 'PLATE_Z_BALLS', 'CUMULATIVE_PI
                  'BAT_PITCHER_MATCHUP', 'INNING_PRESSURE']]

y_FB = cleaned_data['FB']  # Fastball target

# Split the data into training and testing sets
X_train, X_test, y_train_FB, y_test_FB = train_test_split(X, y_FB, test_size=0.2, ran

# Train the Random Forest model with the new features
rf_model = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train_FB)

# Make predictions and evaluate
pred_FB = rf_model.predict(X_test)
mse_FB = mean_squared_error(y_test_FB, pred_FB)
print(f'MSE for Fastball prediction after feature engineering: {mse_FB}')
```

```
MSE for Fastball prediction after feature engineering: 0.1962405277967243
```

In [40]:
```
# Define the feature set with engineered features for Breaking Ball prediction
y_BB = cleaned_data['BB']  # Breaking Ball target

# Split the data into training and testing sets
X_train, X_test, y_train_BB, y_test_BB = train_test_split(X, y_BB, test_size=0.2, ran

# Train the Random Forest model with the new features
rf_model.fit(X_train, y_train_BB)

# Make predictions and evaluate
pred_BB = rf_model.predict(X_test)
```

```
mse_BB = mean_squared_error(y_test_BB, pred_BB)
print(f'MSE for Breaking Ball prediction after feature engineering: {mse_BB}')
```

```
MSE for Breaking Ball prediction after feature engineering: 0.1799481003198426
```

2024 Predictions

The model achieved the following evaluation results:

- MSE for Fastball prediction: 0.20203
- MSE for Breaking Ball prediction: 0.17925
- MSE for Off-Speed prediction: 0.09318

In [1]:
```
# Load the historical data (assuming data.csv contains data from 2020-2023)
historical_data = pd.read_csv('data.csv')

print(historical_data.columns)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 # Load the historical data (assuming data.csv contains data from 2020-2023)
----> 2 historical_data = pd.read_csv('data.csv')
      4 print(historical_data.columns)

NameError: name 'pd' is not defined
```

In [42]:
```
# 1. CUMULATIVE_PITCH_COUNT: Calculate the cumulative number of pitches per game
historical_data['CUMULATIVE_PITCH_COUNT'] = historical_data.groupby('GAME_PK')['PITCH

# 2. BAT_PITCHER_MATCHUP: Combine BAT_SIDE and THROW_SIDE to create a matchup feature
historical_data['BAT_PITCHER_MATCHUP'] = historical_data['BAT_SIDE'] + "_" + historic

# 3. INNING_PRESSURE: Estimate inning pressure based on the inning and score differen
# We assume pressure is higher in later innings and when the score is close
historical_data['SCORE_DIFFERENCE'] = abs(historical_data['HOME_SCORE'] - historical_
historical_data['INNING_PRESSURE'] = historical_data['INNING'] / (1 + historical_data
```

In [43]:
```
# Convert relevant columns to numeric, coercing any errors
numeric_columns = ['PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES', 'INNING',
                   'CUMULATIVE_PITCH_COUNT', 'INNING_PRESSURE']

# Ensure these columns are numeric, converting where necessary
historical_data[numeric_columns] = historical_data[numeric_columns].apply(pd.to_numer

# Now calculate averages for each year, ignoring non-numeric or invalid data
historical_averages = historical_data.groupby('GAME_YEAR')[numeric_columns].mean()

# Display the historical averages
print(historical_averages)
```

```
            PLATE_X   PLATE_Z     BALLS   STRIKES    INNING  \
GAME_YEAR
2021       0.044819  2.269163  0.900848  0.890472  4.843375
2022       0.037257  2.284095  0.884241  0.892525  4.867481
2023       0.032780  2.278574  0.886415  0.899395  4.882166

            CUMULATIVE_PITCH_COUNT  INNING_PRESSURE
GAME_YEAR
2021                    232.542990         2.128805
2022                    272.606628         2.176164
2023                    317.810417         2.126566
```

Estimating 2024 values

In [44]:
```
# Estimate 2024 values by taking the mean of the historical averages
estimated_2024_values = historical_averages.mean()

# Display the estimated values for 2024
print(estimated_2024_values)
```

```
PLATE_X                   0.038286
PLATE_Z                   2.277277
BALLS                     0.890501
STRIKES                   0.894131
INNING                    4.864341
CUMULATIVE_PITCH_COUNT  274.320012
INNING_PRESSURE           2.143845
dtype: float64
```

In [45]:
```
# Assuming you already have your training data (X_train, y_train_FB, etc.)
from sklearn.ensemble import RandomForestRegressor

# Retrain the models if they were not saved
rf_model_FB = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model_FB.fit(X_train, y_train_FB)

rf_model_BB = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model_BB.fit(X_train, y_train_BB)

rf_model_OS = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
rf_model_OS.fit(X_train, y_train_OS)
```

Out[45]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=10, n_estimators=200, random_state=42)
```

In [46]:
```
# Check the feature names that were used during training
print(rf_model_FB.feature_names_in_)
```

```
['PLATE_X' 'PLATE_Z' 'BALLS' 'STRIKES' 'INNING' 'PLATE_X_STRIKES'
 'PLATE_X_BALLS' 'PLATE_Z_STRIKES' 'PLATE_Z_BALLS'
 'CUMULATIVE_PITCH_COUNT' 'BAT_PITCHER_MATCHUP' 'INNING_PRESSURE']
```

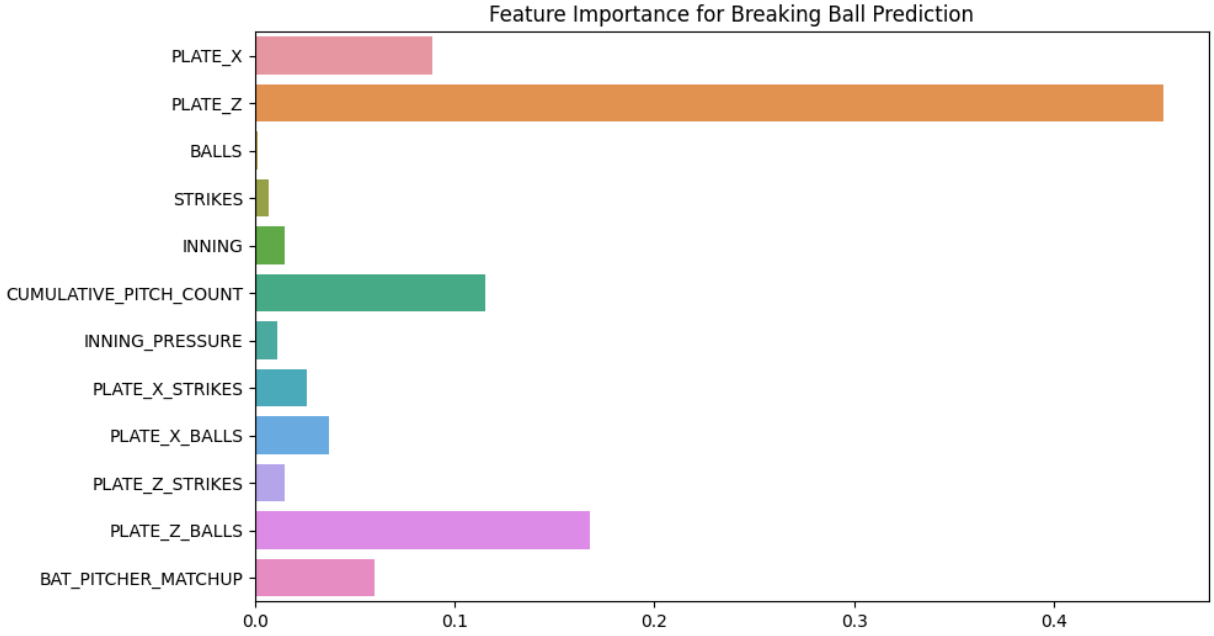
In [57]:
```
# Check the feature names that were used during training
print(rf_model_FB.feature_names_in_)
```

```
['PLATE_X' 'PLATE_Z' 'BALLS' 'STRIKES' 'INNING' 'PLATE_X_STRIKES'
 'PLATE_X_BALLS' 'PLATE_Z_STRIKES' 'PLATE_Z_BALLS'
 'CUMULATIVE_PITCH_COUNT' 'BAT_PITCHER_MATCHUP' 'INNING_PRESSURE']
```

In [101…
```
# Plot feature importances for BB
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=feature_importances_BB, y=features_2024)
plt.title('Feature Importance for Breaking Ball Prediction')
plt.show()
```

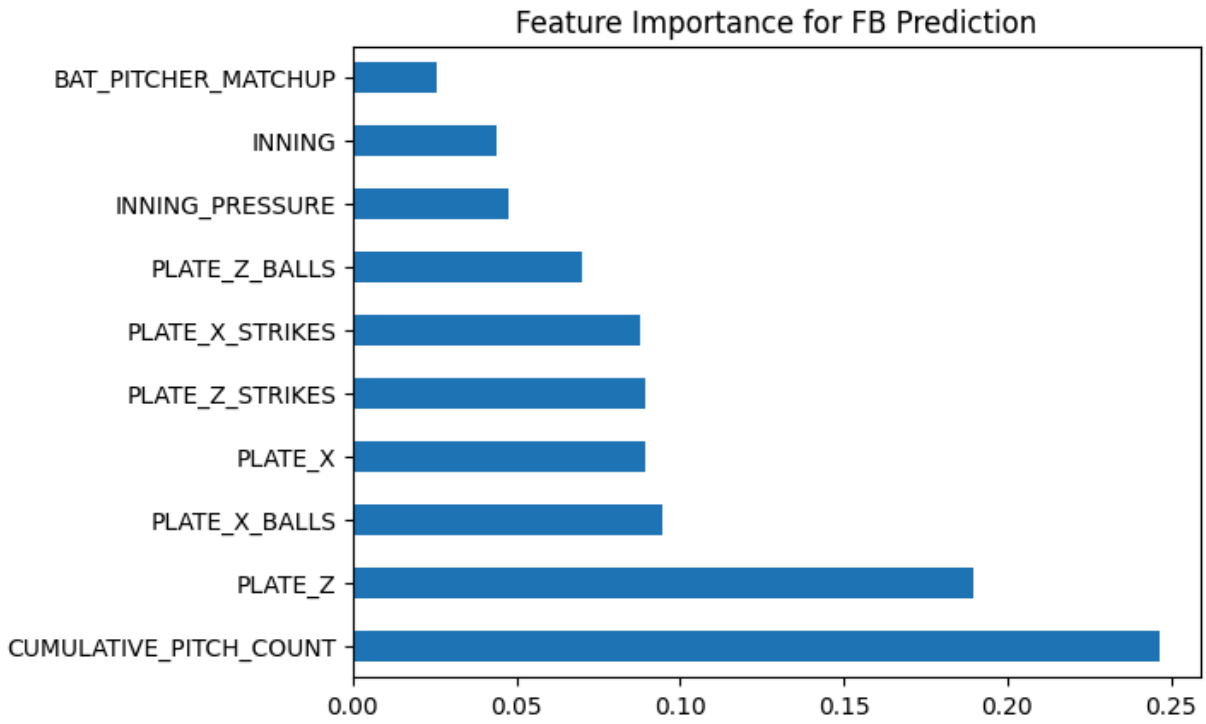
```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: u
nique with argument that is not not a Series, Index, ExtensionArray, or np.ndarray is
deprecated and will raise in a future version.
  order = pd.unique(vector)
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [88]:

```
# Step 4: Feature Selection
# Define features and target for FB prediction
features = ['PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES', 'INNING',
            'CUMULATIVE_PITCH_COUNT', 'INNING_PRESSURE',
            'PLATE_X_STRIKES', 'PLATE_X_BALLS', 'PLATE_Z_STRIKES',
            'PLATE_Z_BALLS', 'BAT_PITCHER_MATCHUP']
target_FB = 'FB'

# Check if all features are present
missing_features = [feature for feature in features if feature not in data.columns]
if missing_features:
    print(f"Missing features for FB prediction: {missing_features}")
else:
    X = data[features]
    y_FB = data[target_FB]

    # Fit a Random Forest to evaluate feature importance
    rf = RandomForestRegressor()
    rf.fit(X, y_FB)

    # Display feature importance
    importances = rf.feature_importances_
    feature_importance = pd.Series(importances, index=features)
```

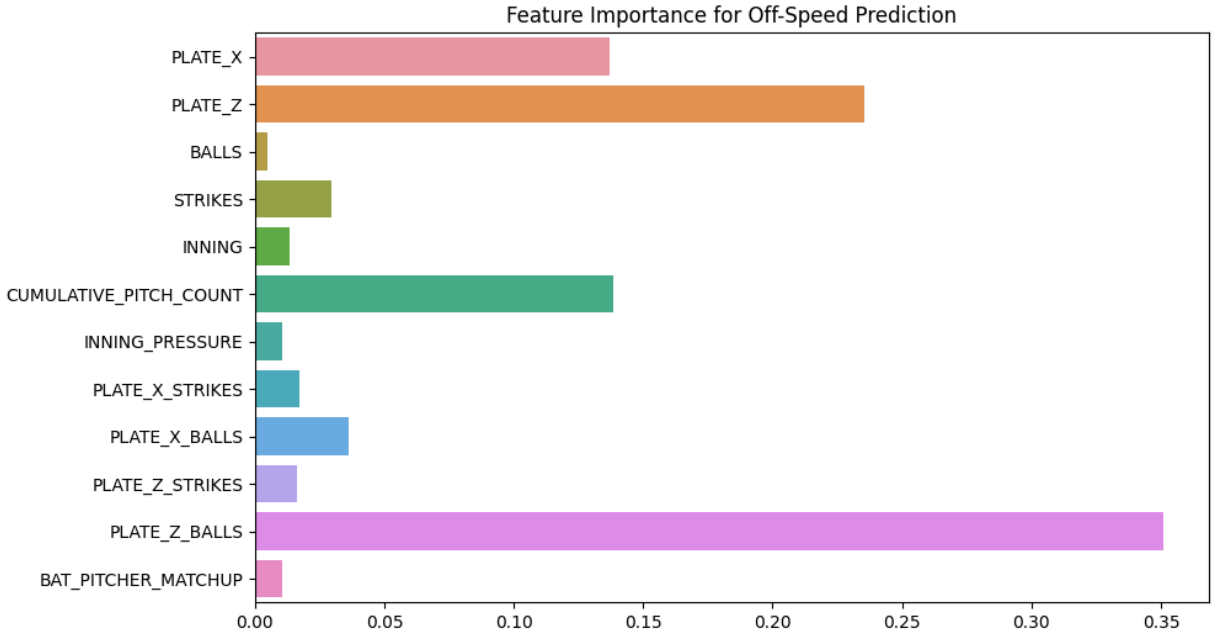
```
    feature_importance.nlargest(10).plot(kind='barh')
    plt.title('Feature Importance for FB Prediction')
    plt.show()
```



In [111…

```
# Plot feature importances for OS
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances_OS, y=features_2024)
plt.title('Feature Importance for Off-Speed Prediction')
plt.show()
```

```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: u
nique with argument that is not not a Series, Index, ExtensionArray, or np.ndarray is
deprecated and will raise in a future version.
  order = pd.unique(vector)
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [102…

```
# Create the 2024 data for prediction using the estimated values
# Create the 2024 data for prediction using the estimated values
data_2024 = pd.DataFrame({
    'BATTER_ID': predictions_data['BATTER_ID'],
    'PLAYER_NAME': predictions_data['PLAYER_NAME'],
    'PLATE_X': [estimated_2024_values['PLATE_X']] * len(predictions_data),  # This sh
    'PLATE_Z': [estimated_2024_values['PLATE_Z']] * len(predictions_data),  # Same he
    'BALLS': [estimated_2024_values['BALLS']] * len(predictions_data),
    'STRIKES': [estimated_2024_values['STRIKES']] * len(predictions_data),
    'INNING': [estimated_2024_values['INNING']] * len(predictions_data),
    'CUMULATIVE_PITCH_COUNT': [0] * len(predictions_data),  # Default or average valu
    'INNING_PRESSURE': [0] * len(predictions_data),  # Default or average value
    'BAT_PITCHER_MATCHUP': [0] * len(predictions_data),  # Placeholder values
})

def calculate_pitch_count_for_player(player_id):
    # Calculate cumulative pitch count based on historical data for the given player_
    # This example assumes you have a DataFrame called 'historical_data' with all nec
    player_data = historical_data[historical_data['BATTER_ID'] == player_id]
    return len(player_data)  # Number of pitches this player has faced (or similar me

def calculate_inning_pressure_for_player(player_id):
    # Calculate inning pressure based on the player's past performance
    player_data = historical_data[historical_data['BATTER_ID'] == player_id]
    # Example logic: average inning pressure based on innings faced
    return player_data['INNING_PRESSURE'].mean() if not player_data.empty else 0

def determine_matchup(player_id):
    # Determine the BAT_PITCHER_MATCHUP based on historical data or a static rule
    # For example, you could return a value based on a mapping of sides
    # This is a placeholder implementation
    player_data = historical_data[historical_data['BATTER_ID'] == player_id]
    if not player_data.empty:
        return player_data['BAT_PITCHER_MATCHUP'].iloc[0]  # Return the first matchup
    return 0  # Default if no data is found


# Now ensure the features are set for each player correctly
data_2024['CUMULATIVE_PITCH_COUNT'] = [calculate_pitch_count_for_player(player_id) fo
data_2024['INNING_PRESSURE'] = [calculate_inning_pressure_for_player(player_id) for p
data_2024['BAT_PITCHER_MATCHUP'] = [determine_matchup(player_id) for player_id in dat
```

In [103…

```
# Encode the BAT_PITCHER_MATCHUP in your training dataset
data['BAT_PITCHER_MATCHUP'] = data['BAT_SIDE'].astype(str) + "_" + data['THROW_SIDE']

# Use Label Encoding for BAT_PITCHER_MATCHUP
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
data['BAT_PITCHER_MATCHUP'] = label_encoder.fit_transform(data['BAT_PITCHER_MATCHUP']

# Ensure to apply the same encoding to the 2024 data
data_2024['BAT_PITCHER_MATCHUP'] = label_encoder.transform(data_2024['BAT_PITCHER_MAT
```

In [106…

```
# Ensure interaction features are calculated
data_2024['PLATE_X_STRIKES'] = data_2024['PLATE_X'] * data_2024['STRIKES']
data_2024['PLATE_X_BALLS'] = data_2024['PLATE_X'] * data_2024['BALLS']
data_2024['PLATE_Z_STRIKES'] = data_2024['PLATE_Z'] * data_2024['STRIKES']
data_2024['PLATE_Z_BALLS'] = data_2024['PLATE_Z'] * data_2024['BALLS']

# Now align the features for Fastball prediction
data_2024_FB_aligned = data_2024[training_features_FB]  # Ensure these features are n

# Now make predictions for Fastball
data_2024['PITCH_TYPE_FB'] = rf_model_FB.predict(data_2024_FB_aligned)

# Repeat for Breaking Ball and Off-Speed if necessary
data_2024_BB_aligned = data_2024[training_features_BB]
data_2024['PITCH_TYPE_BB'] = rf_model_BB.predict(data_2024_BB_aligned)

data_2024_OS_aligned = data_2024[training_features_OS]
data_2024['PITCH_TYPE_OS'] = rf_model_OS.predict(data_2024_OS_aligned)

# Display results
print(data_2024[['BATTER_ID', 'PLAYER_NAME', 'PITCH_TYPE_FB', 'PITCH_TYPE_BB', 'PITCH
```

	BATTER_ID	PLAYER_NAME	PITCH_TYPE_FB	PITCH_TYPE_BB	\
0	444482	Peralta, David	0.426327	0.304449	
1	453568	Blackmon, Charlie	0.465806	0.364039	
2	456781	Solano, Donovan	0.509121	0.304213	
3	457705	McCutchen, Andrew	0.512806	0.304213	
4	457759	Turner, Justin	0.508897	0.350766	
5	467793	Santana, Carlos	0.430600	0.304449	
6	500743	Rojas, Miguel	0.508897	0.345785	
7	502054	Pham, Tommy	0.512806	0.304213	
8	502110	Martinez, J.D.	0.512806	0.304213	
9	502671	Goldschmidt, Paul	0.508897	0.350766	
10	514888	Altuve, Jose	0.508897	0.350766	
11	516782	Marte, Starling	0.512284	0.345785	
12	518595	d'Arnaud, Travis	0.511345	0.344087	
13	518692	Freeman, Freddie	0.430600	0.304449	
14	518792	Heyward, Jason	0.430546	0.304449	
15	519203	Rizzo, Anthony	0.430600	0.304449	
16	519317	Stanton, Giancarlo	0.512806	0.304213	
17	521692	Perez, Salvador	0.508897	0.350766	
18	542303	Ozuna, Marcell	0.512284	0.345785	
19	543257	Grossman, Robbie	0.430600	0.304449	
20	543309	Higashioka, Kyle	0.503151	0.397760	
21	543760	Semien, Marcus	0.508897	0.350766	
22	543807	Springer, George	0.508897	0.350766	
23	543877	Vázquez, Christian	0.509121	0.304213	
24	545121	Vargas, Ildemaro	0.510370	0.304213	
25	545341	Grichuk, Randal	0.508897	0.350766	
26	547180	Harper, Bryce	0.465806	0.364039	
27	553869	Díaz, Elias	0.509121	0.304213	
28	553993	Suárez, Eugenio	0.508897	0.350766	
29	570482	Urshela, Gio	0.509121	0.304213	
30	571448	Arenado, Nolan	0.508897	0.350766	
31	571745	Haniger, Mitch	0.509221	0.344087	
32	571771	Hernández, Enrique	0.512806	0.304213	
33	571970	Muncy, Max	0.430600	0.304449	
34	572138	Singleton, Jon	0.438403	0.304449	
35	572191	Taylor, Michael A.	0.508897	0.345785	
36	572233	Walker, Christian	0.508897	0.350766	
37	573262	Yastrzemski, Mike	0.465806	0.364039	
38	575929	Contreras, Willson	0.508897	0.350766	
39	578428	Iglesias, Jose	0.511345	0.344087	
40	592192	Canha, Mark	0.508897	0.350766	
41	592206	Castellanos, Nick	0.508897	0.350766	
42	592273	Drury, Brandon	0.509221	0.344087	
43	592450	Judge, Aaron	0.512806	0.304213	
44	592518	Machado, Manny	0.512806	0.304213	
45	592626	Pederson, Joc	0.426327	0.304449	
46	592663	Realmuto, J.T.	0.512806	0.304213	
47	592669	Renfroe, Hunter	0.512806	0.304213	
48	592696	Rosario, Eddie	0.463691	0.364039	
49	592885	Yelich, Christian	0.430600	0.304449	
50	593160	Merrifield, Whit	0.508897	0.350766	
51	593428	Bogaerts, Xander	0.512806	0.304213	
52	593871	Polanco, Jorge	0.430600	0.304449	
53	594807	Duvall, Adam	0.509221	0.344087	
54	595281	Kiermaier, Kevin	0.426886	0.304449	
55	595777	Profar, Jurickson	0.512806	0.304213	
56	595879	Báez, Javier	0.508897	0.350766	
57	595909	Cave, Jake	0.430546	0.304449	
58	596019	Lindor, Francisco	0.512806	0.304213	
59	596142	Sánchez, Gary	0.509121	0.304213	
60	596146	Kepler, Max	0.426327	0.304449	
61	600869	Candelario, Jeimer	0.430600	0.304449	
62	602104	Urías, Ramón	0.509121	0.304213	

63	605137	Bell, Josh	0.430600	0.304449
64	605141	Betts, Mookie	0.508897	0.350766
65	605170	Caratini, Victor	0.512620	0.304213
66	606115	Arcia, Orlando	0.506432	0.344087
67	606192	Hernández, Teoscar	0.508897	0.350766
68	606466	Marte, Ketel	0.430600	0.304449
69	607043	Nimmo, Brandon	0.465806	0.364039
70	607208	Turner, Trea	0.512806	0.304213
71	607680	Pillar, Kevin	0.510370	0.304213
72	607732	Stallings, Jacob	0.506770	0.344087
73	608070	Ramírez, José	0.512806	0.304213
74	608324	Bregman, Alex	0.508897	0.350766
75	608336	Gallo, Joey	0.430600	0.304449
76	608348	Kelly, Carson	0.510512	0.304213
77	608369	Seager, Corey	0.426327	0.304449
78	608385	Winker, Jesse	0.426916	0.304449
79	608701	Refsnyder, Rob	0.510370	0.304213
80	608841	Meneses, Joey	0.510540	0.344087
81	609280	Andujar, Miguel	0.510280	0.304213
82	621020	Swanson, Dansby	0.508897	0.350766
83	621028	Newman, Kevin	0.509064	0.344087
84	621035	Taylor, Chris	0.508897	0.350766
85	621043	Correa, Carlos	0.508897	0.350766
86	621438	Taylor, Tyrone	0.510540	0.344087
87	621439	Buxton, Byron	0.509834	0.344087
88	621493	Ward, Taylor	0.509221	0.344087
89	621566	Olson, Matt	0.430600	0.304449
90	622534	Margot, Manuel	0.509221	0.344087
91	623993	Santander, Anthony	0.430600	0.304449
92	624413	Alonso, Pete	0.512806	0.304213
93	624424	Conforto, Michael	0.463624	0.367091
94	624428	Frazier, Adam	0.430600	0.304449
95	624503	Gordon, Nick	0.430546	0.304449
96	624585	Soler, Jorge	0.508897	0.350766
97	624641	Sosa, Edmundo	0.506634	0.384727
98	630105	Cronenworth, Jake	0.465806	0.364039
99	641343	Bauers, Jake	0.430546	0.304449
100	641355	Bellinger, Cody	0.430600	0.304449
101	641487	Crawford, J.P.	0.430600	0.304449
102	641584	Fraley, Jake	0.429367	0.304449
103	641598	Garver, Mitch	0.506432	0.344087
104	641680	Heim, Jonah	0.426916	0.304449
105	641857	McMahon, Ryan	0.465806	0.364039
106	641933	O'Neill, Tyler	0.509221	0.344087
107	642086	Smith, Dominic	0.426916	0.304449
108	642133	Tellez, Rowdy	0.428485	0.304449
109	642350	Siri, Jose	0.503584	0.397356
110	642708	Rosario, Amed	0.512806	0.304213
111	642715	Adames, Willy	0.508897	0.350766
112	642731	Estrada, Thairo	0.509221	0.344087
113	643217	Benintendi, Andrew	0.430600	0.304449
114	643289	Dubón, Mauricio	0.512188	0.304213
115	643376	Jansen, Danny	0.506634	0.387085
116	643396	Kiner-Falefa, Isiah	0.508897	0.350766
117	643446	McNeil, Jeff	0.465806	0.364039
118	643565	Tauchman, Mike	0.430546	0.304449
119	645277	Albies, Ozzie	0.430600	0.304449
120	645302	Robles, Victor	0.512188	0.304213
121	646240	Devers, Rafael	0.465806	0.364039
122	647304	Naylor, Josh	0.463691	0.364039
123	647351	Toro, Abraham	0.430546	0.304449
124	650333	Arraez, Luis	0.430600	0.304449
125	650391	Jiménez, Eloy	0.510512	0.304213
126	650402	Torres, Gleyber	0.512806	0.304213

127	650489	Castro, Willi	0.426916	0.304449
128	650490	Díaz, Yandy	0.508897	0.350766
129	650559	De La Cruz, Bryan	0.509221	0.344087
130	650859	Rengifo, Luis	0.508964	0.304213
131	656305	Chapman, Matt	0.508897	0.350766
132	656555	Hoskins, Rhys	0.509121	0.304213
133	656582	Joe, Connor	0.509221	0.344087
134	656716	McKinstry, Zach	0.429367	0.304449
135	656775	Mullins, Cedric	0.430600	0.304449
136	656811	O'Hearn, Ryan	0.430546	0.304449
137	656896	Rivera, Emmanuel	0.504600	0.397356
138	656941	Schwarber, Kyle	0.430600	0.304449
139	657041	Thomas, Lane	0.512806	0.304213
140	657077	Verdugo, Alex	0.465806	0.364039
141	657136	Wong, Connor	0.510280	0.304213
142	657557	DeJong, Paul	0.509064	0.344087
143	657656	Laureano, Ramón	0.509221	0.344087
144	657757	Sheets, Gavin	0.426849	0.304449
145	660271	Ohtani, Shohei	0.430600	0.304449
146	660644	Bruján, Vidal	0.430455	0.304449
147	660688	Ruiz, Keibert	0.427856	0.304449
148	660821	Sánchez, Jesús	0.426849	0.304449
149	661388	Contreras, William	0.509221	0.344087
150	662139	Varsho, Daulton	0.430600	0.304449
151	663368	Perkins, Blake	0.437320	0.304449
152	663457	Nootbaar, Lars	0.463624	0.367091
153	663527	Nevin, Tyler	0.510280	0.304213
154	663538	Hoerner, Nico	0.513641	0.345785
155	663586	Riley, Austin	0.508897	0.350766
156	663616	Larnach, Trevor	0.430546	0.304449
157	663624	Mountcastle, Ryan	0.508897	0.350766
158	663647	Hayes, Ke'Bryan	0.508897	0.350766
159	663656	Tucker, Kyle	0.430600	0.304449
160	663697	India, Jonathan	0.508897	0.350766
161	663698	Bart, Joey	0.510280	0.304213
162	663728	Raleigh, Cal	0.509121	0.304213
163	663743	Fortes, Nick	0.491075	0.421160
164	663837	Vierling, Matt	0.509001	0.304213
165	663886	Stephenson, Tyler	0.509221	0.344087
166	663898	Rodgers, Brendan	0.509221	0.344087
167	663993	Lowe, Nathaniel	0.430600	0.304449
168	664023	Happ, Ian	0.430600	0.304449
169	664034	France, Ty	0.508897	0.350766
170	664040	Lowe, Brandon	0.428485	0.304449
171	664056	Bader, Harrison	0.509064	0.344087
172	664238	Moore, Dylan	0.506432	0.344087
173	664728	Isbel, Kyle	0.430546	0.304449
174	664761	Bohm, Alec	0.512806	0.304213
175	664774	Wade Jr., LaMonte	0.426916	0.304449
176	664913	Brown, Seth	0.426916	0.304449
177	664983	McCarthy, Jake	0.430546	0.304449
178	665161	Peña, Jeremy	0.509221	0.344087
179	665487	Tatis Jr., Fernando	0.509121	0.304213
180	665489	Guerrero Jr., Vladimir	0.508897	0.350766
181	665742	Soto, Juan	0.465806	0.364039
182	665750	Taveras, Leody	0.426849	0.304449
183	665804	Amaya, Miguel	0.511977	0.304213
184	665828	Cabrera, Oswaldo	0.429533	0.304449
185	665833	Cruz, Oneil	0.430455	0.304449
186	665862	Chisholm Jr., Jazz	0.426916	0.304449
187	665926	Giménez, Andrés	0.461533	0.364039
188	666023	Fermin, Freddy	0.490656	0.422901
189	666134	Jones, Nolan	0.430546	0.304449
190	666139	Lowe, Josh	0.430546	0.304449

191	666149	Fitzgerald, Tyler	0.487849	0.423813
192	666152	Hamilton, David	0.439945	0.304449
193	666158	Lux, Gavin	0.430546	0.304449
194	666160	Moniak, Mickey	0.430455	0.304449
195	666163	Rortvedt, Ben	0.437320	0.304449
196	666176	Adell, Jo	0.491075	0.421691
197	666181	Benson, Will	0.430455	0.304449
198	666182	Bichette, Bo	0.508897	0.350766
199	666185	Carlson, Dylan	0.430600	0.304449
200	666310	Naylor, Bo	0.430455	0.304449
201	666397	Julien, Edouard	0.430455	0.304449
202	666624	Morel, Christopher	0.506432	0.345061
203	666969	García, Adolis	0.512806	0.304213
204	666971	Gurriel Jr., Lourdes	0.508897	0.350766
205	667670	Rooker, Brent	0.506432	0.371717
206	668227	Arozarena, Randy	0.508897	0.350766
207	668670	Rogers, Jake	0.491075	0.421691
208	668709	Bleday, JJ	0.430546	0.304449
209	668715	Steer, Spencer	0.512188	0.304213
210	668804	Reynolds, Bryan	0.430600	0.304449
211	668901	Vientos, Mark	0.510280	0.304213
212	668904	Lewis, Royce	0.510280	0.304213
213	668930	Turang, Brice	0.430455	0.304449
214	668939	Rutschman, Adley	0.509121	0.304213
215	668942	Rojas, Josh	0.430600	0.304449
216	669004	Melendez, MJ	0.426916	0.304449
217	669016	Marsh, Brandon	0.426916	0.304449
218	669127	Langeliers, Shea	0.498354	0.403794
219	669134	Campusano, Luis	0.490656	0.422901
220	669221	Murphy, Sean	0.508897	0.350766
221	669222	Senzel, Nick	0.506432	0.381539
222	669224	Wells, Austin	0.438403	0.304449
223	669257	Smith, Will	0.508897	0.350766
224	669261	Suwinski, Jack	0.426886	0.304449
225	669289	Espinal, Santiago	0.509001	0.304213
226	669304	Miranda, Jose	0.492711	0.406217
227	669357	Gorman, Nolan	0.430546	0.304449
228	669364	Edwards, Xavier	0.438403	0.304449
229	669394	Burger, Jake	0.504350	0.397356
230	669701	Smith, Josh	0.430455	0.304449
231	669707	Triolo, Jared	0.510638	0.304213
232	669911	Toglia, Michael	0.510280	0.304213
233	670032	Lopez, Nicky	0.463691	0.364039
234	670042	Raley, Luke	0.465462	0.371973
235	670242	Wallner, Matt	0.465462	0.372185
236	670541	Alvarez, Yordan	0.430600	0.304449
237	670623	Paredes, Isaac	0.509121	0.304213
238	670764	Walls, Taylor	0.508964	0.304213
239	670770	Friedl, TJ	0.430546	0.304449
240	671056	Herrera, Iván	0.491108	0.422518
241	671218	Ramos, Heliot	0.511977	0.304213
242	671277	García Jr., Luis	0.426849	0.304449
243	671289	Freeman, Tyler	0.510280	0.304213
244	671732	Butler, Lawrence	0.438403	0.304449
245	671739	Harris II, Michael	0.429367	0.304449
246	672275	Bailey, Patrick	0.510280	0.304213
247	672279	Siani, Michael	0.442552	0.304449
248	672284	Kelenic, Jarred	0.426886	0.304449
249	672386	Kirk, Alejandro	0.509221	0.344087
250	672515	Moreno, Gabriel	0.490723	0.422010
251	672580	Garcia, Maikel	0.491075	0.421160
252	672640	Lopez, Otto	0.494694	0.422518
253	672695	Perdomo, Geraldo	0.506671	0.304213
254	672820	Sosa, Lenyn	0.490430	0.422901

255	673237	Diaz, Yainer	0.510280	0.304213
256	673357	Robert Jr., Luis	0.509221	0.344087
257	673490	Kim, Ha-Seong	0.512806	0.304213
258	673548	Suzuki, Seiya	0.509221	0.344087
259	676059	Westburg, Jordan	0.510638	0.304213
260	676356	DeLuca, Jonny	0.513355	0.304213
261	676391	Clement, Ernie	0.490225	0.423172
262	676475	Burleson, Alec	0.430455	0.304449
263	676609	Caballero, José	0.490225	0.423172
264	676694	Meyers, Jake	0.510842	0.304213
265	676801	McCormick, Chas	0.509121	0.304213
266	676914	Schneider, Davis	0.511977	0.304213
267	677587	Rocchio, Brayan	0.438403	0.304449
268	677594	Rodríguez, Julio	0.509221	0.344087
269	677649	Duran, Ezequiel	0.510370	0.304213
270	677800	Abreu, Wilyer	0.438403	0.304449
271	677951	Witt Jr., Bobby	0.513641	0.345785
272	678009	Meadows, Parker	0.438403	0.304449
273	678246	Vargas, Miguel	0.490483	0.423172
274	678662	Tovar, Ezequiel	0.510370	0.304213
275	678882	Rafaela, Ceddanne	0.487593	0.423813
276	679032	Rojas, Johan	0.511977	0.304213
277	679529	Torkelson, Spencer	0.509221	0.344087
278	680700	Palacios, Richie	0.431058	0.304449
279	680757	Kwan, Steven	0.426327	0.304449
280	680776	Duran, Jarren	0.430546	0.304449
281	680777	Jeffers, Ryan	0.510540	0.344087
282	680869	Gelof, Zack	0.490225	0.423172
283	680977	Donovan, Brendan	0.430546	0.304449
284	681082	Stott, Bryson	0.426916	0.304449
285	681146	Bride, Jonah	0.490483	0.423172
286	681297	Cowser, Colton	0.438403	0.304449
287	681351	O'Hoppe, Logan	0.490656	0.422901
288	681481	Carpenter, Kerry	0.430546	0.304449
289	681807	Fry, David	0.511977	0.304213
290	682626	Alvarez, Francisco	0.510280	0.304213
291	682829	De La Cruz, Elly	0.430455	0.304449
292	682928	Abrams, CJ	0.430546	0.304449
293	682985	Greene, Riley	0.465135	0.367984
294	682998	Carroll, Corbin	0.466314	0.370661
295	683002	Henderson, Gunnar	0.430546	0.304449
296	683011	Volpe, Anthony	0.492711	0.407961
297	683734	Vaughn, Andrew	0.512806	0.304213
298	683737	Busch, Michael	0.438403	0.304449
299	686217	Frelick, Sal	0.431416	0.304449
300	686469	Pasquantino, Vinnie	0.430455	0.304449
301	686668	Doyle, Brenton	0.490723	0.422589
302	686676	Lee, Korey	0.490921	0.422518
303	686681	Massey, Michael	0.430546	0.304449
304	686823	Brennan, Will	0.430455	0.304449
305	687263	Neto, Zach	0.490483	0.423172
306	687401	Ortiz, Joey	0.515568	0.304213
307	687462	Horwitz, Spencer	0.437766	0.304449
308	691026	Winn, Masyn	0.511977	0.304213
309	691718	Crow-Armstrong, Pete	0.443233	0.304449
310	693304	Gonzales, Nick	0.512408	0.304213
311	694384	Schanuel, Nolan	0.438403	0.304449
312	696285	Young, Jacob	0.490921	0.422518
313	807799	Yoshida, Masataka	0.430546	0.304449

	PITCH_TYPE_OS
0	0.140889
1	0.060615
2	0.166813

3	0.166813
4	0.106422
5	0.142915
6	0.099449
7	0.166813
8	0.166813
9	0.106422
10	0.106422
11	0.080425
12	0.067361
13	0.142915
14	0.140889
15	0.142915
16	0.166813
17	0.106422
18	0.080425
19	0.142915
20	0.066688
21	0.106422
22	0.106422
23	0.166813
24	0.166813
25	0.106422
26	0.060615
27	0.166813
28	0.106422
29	0.166813
30	0.106422
31	0.071330
32	0.166813
33	0.142915
34	0.140398
35	0.109345
36	0.106422
37	0.060615
38	0.106422
39	0.067361
40	0.106422
41	0.106422
42	0.068973
43	0.166813
44	0.166813
45	0.140889
46	0.166813
47	0.166813
48	0.060615
49	0.142915
50	0.106422
51	0.166813
52	0.142915
53	0.071330
54	0.140889
55	0.166813
56	0.106422
57	0.140889
58	0.166813
59	0.166813
60	0.142915
61	0.142915
62	0.166813
63	0.142915
64	0.106422
65	0.166813
66	0.066971

67	0.106422
68	0.142915
69	0.060615
70	0.166813
71	0.166813
72	0.071980
73	0.166813
74	0.106422
75	0.142915
76	0.166813
77	0.142915
78	0.140889
79	0.166813
80	0.066971
81	0.166813
82	0.106422
83	0.067361
84	0.106422
85	0.106422
86	0.066971
87	0.067361
88	0.075897
89	0.142915
90	0.068973
91	0.142915
92	0.166813
93	0.057489
94	0.142915
95	0.140889
96	0.106422
97	0.071443
98	0.060615
99	0.140889
100	0.142915
101	0.142915
102	0.140889
103	0.066971
104	0.140889
105	0.060615
106	0.071330
107	0.140889
108	0.140889
109	0.066688
110	0.166813
111	0.106422
112	0.068973
113	0.142915
114	0.166813
115	0.066688
116	0.106422
117	0.060615
118	0.140889
119	0.142915
120	0.166813
121	0.060615
122	0.061335
123	0.140889
124	0.142915
125	0.166813
126	0.166813
127	0.140889
128	0.106422
129	0.071330
130	0.166813

131	0.106422
132	0.166813
133	0.071330
134	0.140889
135	0.142915
136	0.140889
137	0.066688
138	0.142915
139	0.166813
140	0.060615
141	0.166813
142	0.067361
143	0.071154
144	0.140889
145	0.142915
146	0.140398
147	0.140889
148	0.140889
149	0.071330
150	0.142915
151	0.140398
152	0.071505
153	0.166545
154	0.076699
155	0.106422
156	0.140889
157	0.106422
158	0.109993
159	0.142915
160	0.106422
161	0.166813
162	0.166813
163	0.066688
164	0.166813
165	0.068973
166	0.068973
167	0.142915
168	0.142915
169	0.106422
170	0.140889
171	0.067361
172	0.066971
173	0.140889
174	0.166813
175	0.140889
176	0.140889
177	0.140889
178	0.068093
179	0.166813
180	0.106422
181	0.060615
182	0.140889
183	0.165225
184	0.140889
185	0.140398
186	0.140889
187	0.060615
188	0.062365
189	0.140889
190	0.140889
191	0.063265
192	0.140398
193	0.140889
194	0.140398

195	0.140398
196	0.066688
197	0.140398
198	0.106422
199	0.142915
200	0.140398
201	0.140398
202	0.066787
203	0.166813
204	0.106422
205	0.066787
206	0.106422
207	0.066688
208	0.140889
209	0.166813
210	0.142915
211	0.165225
212	0.165225
213	0.140398
214	0.166813
215	0.142915
216	0.140889
217	0.140889
218	0.066688
219	0.062365
220	0.106422
221	0.066787
222	0.140398
223	0.106422
224	0.140889
225	0.166813
226	0.066688
227	0.140889
228	0.140398
229	0.066688
230	0.140889
231	0.165225
232	0.165225
233	0.060615
234	0.045975
235	0.046763
236	0.142915
237	0.166813
238	0.166813
239	0.140889
240	0.062884
241	0.165225
242	0.140889
243	0.165225
244	0.140398
245	0.140889
246	0.166813
247	0.140398
248	0.140889
249	0.071330
250	0.065960
251	0.066688
252	0.063155
253	0.166813
254	0.062192
255	0.166545
256	0.068973
257	0.166813
258	0.068973

```
259       0.165225
260       0.165225
261       0.062609
262       0.140398
263       0.062609
264       0.166813
265       0.166813
266       0.165225
267       0.140398
268       0.068973
269       0.166813
270       0.140398
271       0.076699
272       0.140398
273       0.062703
274       0.166813
275       0.062994
276       0.165225
277       0.068973
278       0.140398
279       0.142915
280       0.140889
281       0.066971
282       0.062609
283       0.140889
284       0.140889
285       0.062609
286       0.140398
287       0.062365
288       0.140889
289       0.165225
290       0.166813
291       0.140398
292       0.140889
293       0.065778
294       0.048990
295       0.140889
296       0.066688
297       0.166813
298       0.140398
299       0.140398
300       0.140889
301       0.063623
302       0.062884
303       0.140889
304       0.140398
305       0.062609
306       0.165225
307       0.140398
308       0.165225
309       0.140398
310       0.165225
311       0.140398
312       0.062884
313       0.140889
```

In [110…

```
print("Features for Fastball:", training_features_FB)
print("Features for Breaking Ball:", training_features_BB)
print("Features for Off-Speed:", training_features_OS)
```

```
Features for Fastball: ['PLATE_X' 'PLATE_Z' 'BALLS' 'STRIKES' 'INNING' 'CUMULATIVE_PI
TCH_COUNT'
 'INNING_PRESSURE' 'PLATE_X_STRIKES' 'PLATE_X_BALLS' 'PLATE_Z_STRIKES'
 'PLATE_Z_BALLS' 'BAT_PITCHER_MATCHUP']
Features for Breaking Ball: ['PLATE_X' 'PLATE_Z' 'BALLS' 'STRIKES' 'INNING' 'PLATE_X_
STRIKES'
 'PLATE_X_BALLS' 'PLATE_Z_STRIKES' 'PLATE_Z_BALLS'
 'CUMULATIVE_PITCH_COUNT' 'BAT_PITCHER_MATCHUP' 'INNING_PRESSURE']
Features for Off-Speed: ['PLATE_X' 'PLATE_Z' 'BALLS' 'STRIKES' 'INNING' 'PLATE_X_STRI
KES'
 'PLATE_X_BALLS' 'PLATE_Z_STRIKES' 'PLATE_Z_BALLS'
 'CUMULATIVE_PITCH_COUNT' 'BAT_PITCHER_MATCHUP' 'INNING_PRESSURE']
```

In [108…

```
print("NaN counts in Fastball aligned data:", data_2024_FB_aligned.isnull().sum())
print("NaN counts in Breaking Ball aligned data:", data_2024_BB_aligned.isnull().sum(
print("NaN counts in Off-Speed aligned data:", data_2024_OS_aligned.isnull().sum())
```

```
NaN counts in Fastball aligned data: PLATE_X                  0
PLATE_Z                    0
BALLS                      0
STRIKES                    0
INNING                     0
CUMULATIVE_PITCH_COUNT     0
INNING_PRESSURE            0
PLATE_X_STRIKES            0
PLATE_X_BALLS              0
PLATE_Z_STRIKES            0
PLATE_Z_BALLS              0
BAT_PITCHER_MATCHUP        0
dtype: int64
NaN counts in Breaking Ball aligned data: PLATE_X                   0
PLATE_Z                    0
BALLS                      0
STRIKES                    0
INNING                     0
PLATE_X_STRIKES            0
PLATE_X_BALLS              0
PLATE_Z_STRIKES            0
PLATE_Z_BALLS              0
CUMULATIVE_PITCH_COUNT     0
BAT_PITCHER_MATCHUP        0
INNING_PRESSURE            0
dtype: int64
NaN counts in Off-Speed aligned data: PLATE_X                   0
PLATE_Z                    0
BALLS                      0
STRIKES                    0
INNING                     0
PLATE_X_STRIKES            0
PLATE_X_BALLS              0
PLATE_Z_STRIKES            0
PLATE_Z_BALLS              0
CUMULATIVE_PITCH_COUNT     0
BAT_PITCHER_MATCHUP        0
INNING_PRESSURE            0
dtype: int64
```

In [112…

```
data_2024.to_csv('predicted_pitch_types_2024.csv', index=False)
```

In [118…

```
# Print the columns of data_2024 to see what is available
print(data_2024.columns)

# Based on the output, adjust the selected_columns list
selected_columns = ['BATTER_ID', 'PLAYER_NAME', 'PITCH_TYPE_FB', 'PITCH_TYPE_BB', 'PI
```

```
# Create a new DataFrame with only the selected columns
filtered_data = data_2024[selected_columns]

# Optionally, save this filtered DataFrame to a new CSV file
filtered_data.to_csv('filtered_predictions.csv', index=False)
```

```
Index(['BATTER_ID', 'PLAYER_NAME', 'PLATE_X', 'PLATE_Z', 'BALLS', 'STRIKES',
       'INNING', 'CUMULATIVE_PITCH_COUNT', 'INNING_PRESSURE',
       'BAT_PITCHER_MATCHUP', 'PLATE_X_STRIKES', 'PLATE_X_BALLS',
       'PLATE_Z_STRIKES', 'PLATE_Z_BALLS', 'PITCH_TYPE_FB', 'PITCH_TYPE_BB',
       'PITCH_TYPE_OS'],
      dtype='object')
```

In [117…

```
# Create a new column 'GAME_YEAR' and set it to 2024 for all rows
filtered_data['GAME_YEAR'] = 2024

# Reorder the columns to include 'GAME_YEAR' in the desired position
filtered_data = filtered_data[['BATTER_ID', 'PLAYER_NAME', 'GAME_YEAR', 'PITCH_TYPE_F



# Optionally, save this updated DataFrame to a new CSV file
filtered_data.to_csv('filtered_predictions_with_year.csv', index=False)
```

Limitations

Data Limitations

• The historical data may not capture all variables influencing pitch types, such as recent changes in player performance or pitcher strategies.

• Missing values were handled but may still affect model reliability.

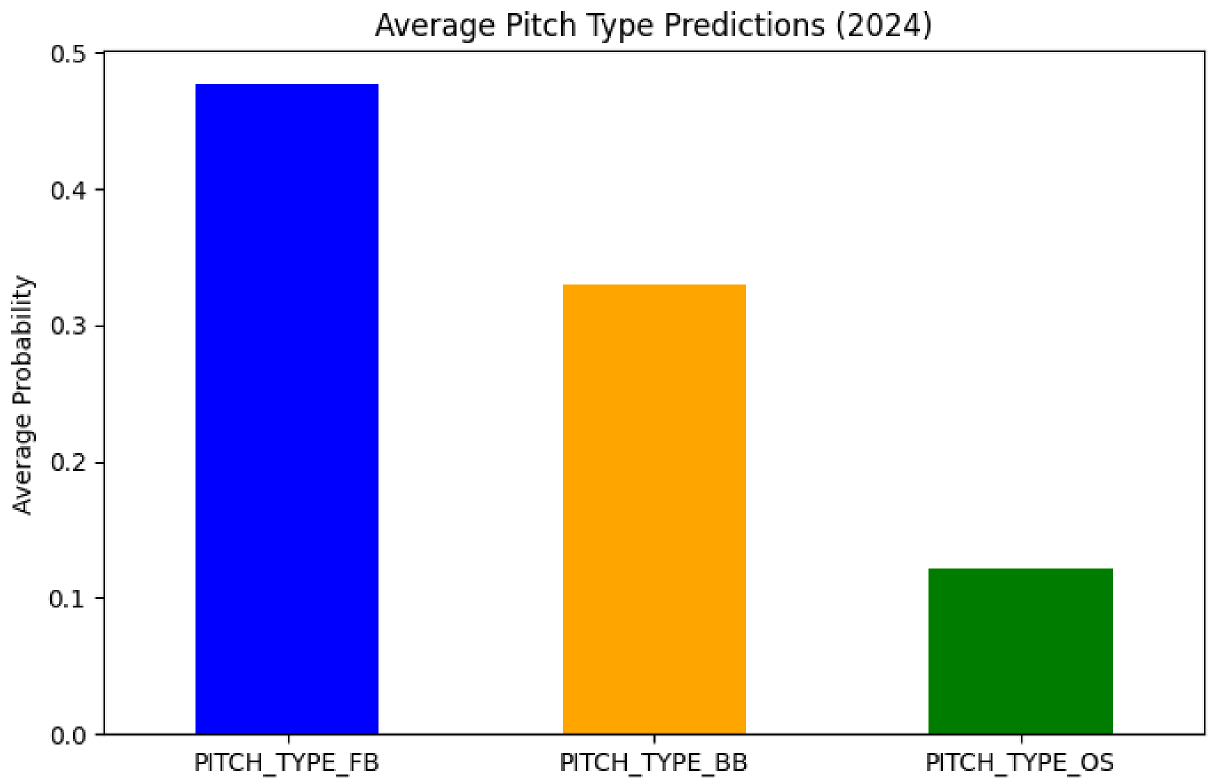
Model Limitations

• The model may overfit the training data, leading to less accurate predictions on unseen data.

• Predictions are probabilistic and do not guarantee specific outcomes in games.

In [116…

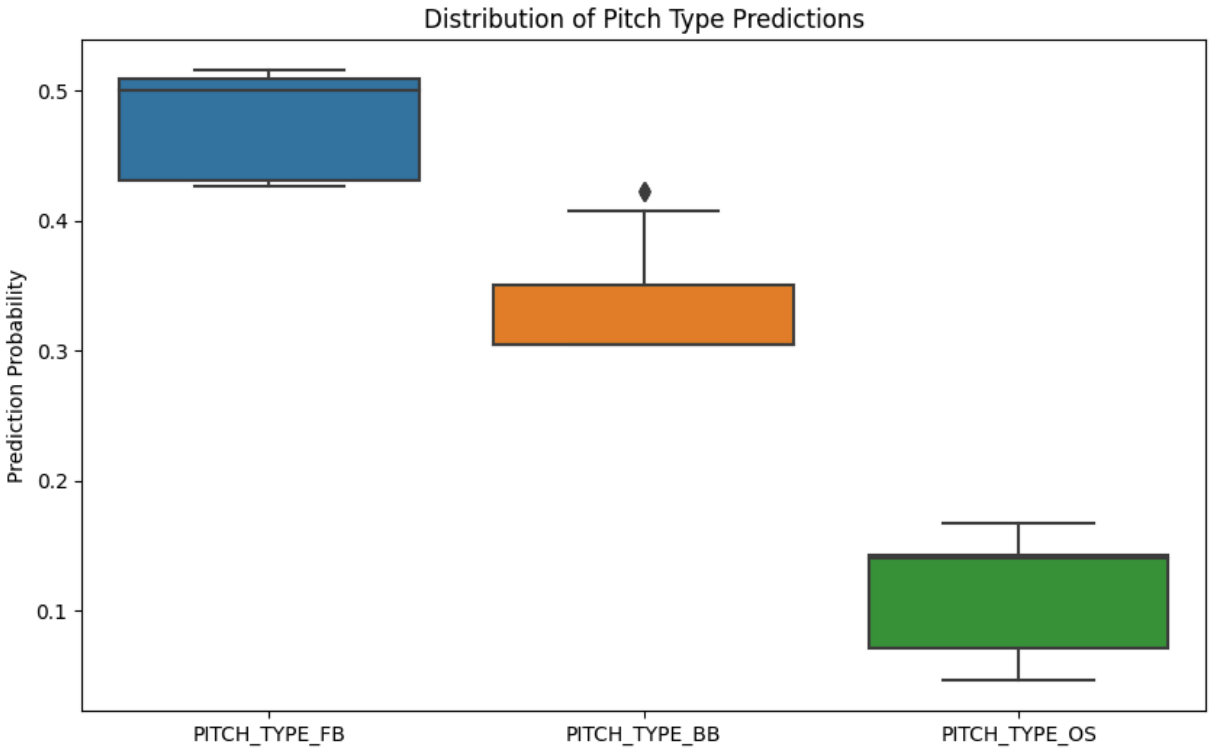
```
 #1. Bar Plot of Average Pitch Predictions
avg_predictions = filtered_data[['PITCH_TYPE_FB', 'PITCH_TYPE_BB', 'PITCH_TYPE_OS']].
plt.figure(figsize=(8, 5))
avg_predictions.plot(kind='bar', color=['blue', 'orange', 'green'])
plt.title('Average Pitch Type Predictions (2024)')
plt.ylabel('Average Probability')
plt.xticks(rotation=0)
plt.show()
```



In [119…

```
# 2. Box Plot of Pitch Predictions
plt.figure(figsize=(10, 6))
sns.boxplot(data=filtered_data[['PITCH_TYPE_FB', 'PITCH_TYPE_BB', 'PITCH_TYPE_OS']])
plt.title('Distribution of Pitch Type Predictions')
plt.ylabel('Prediction Probability')
plt.show()
```

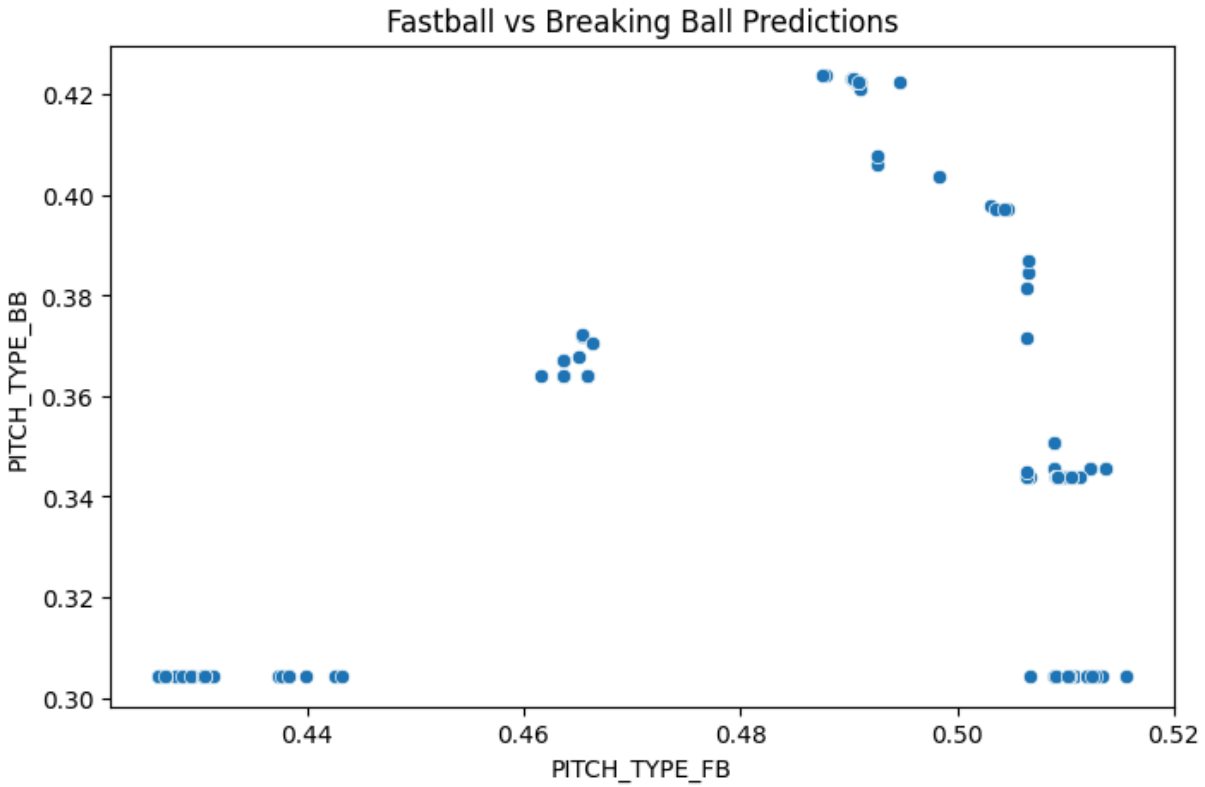
```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [120…

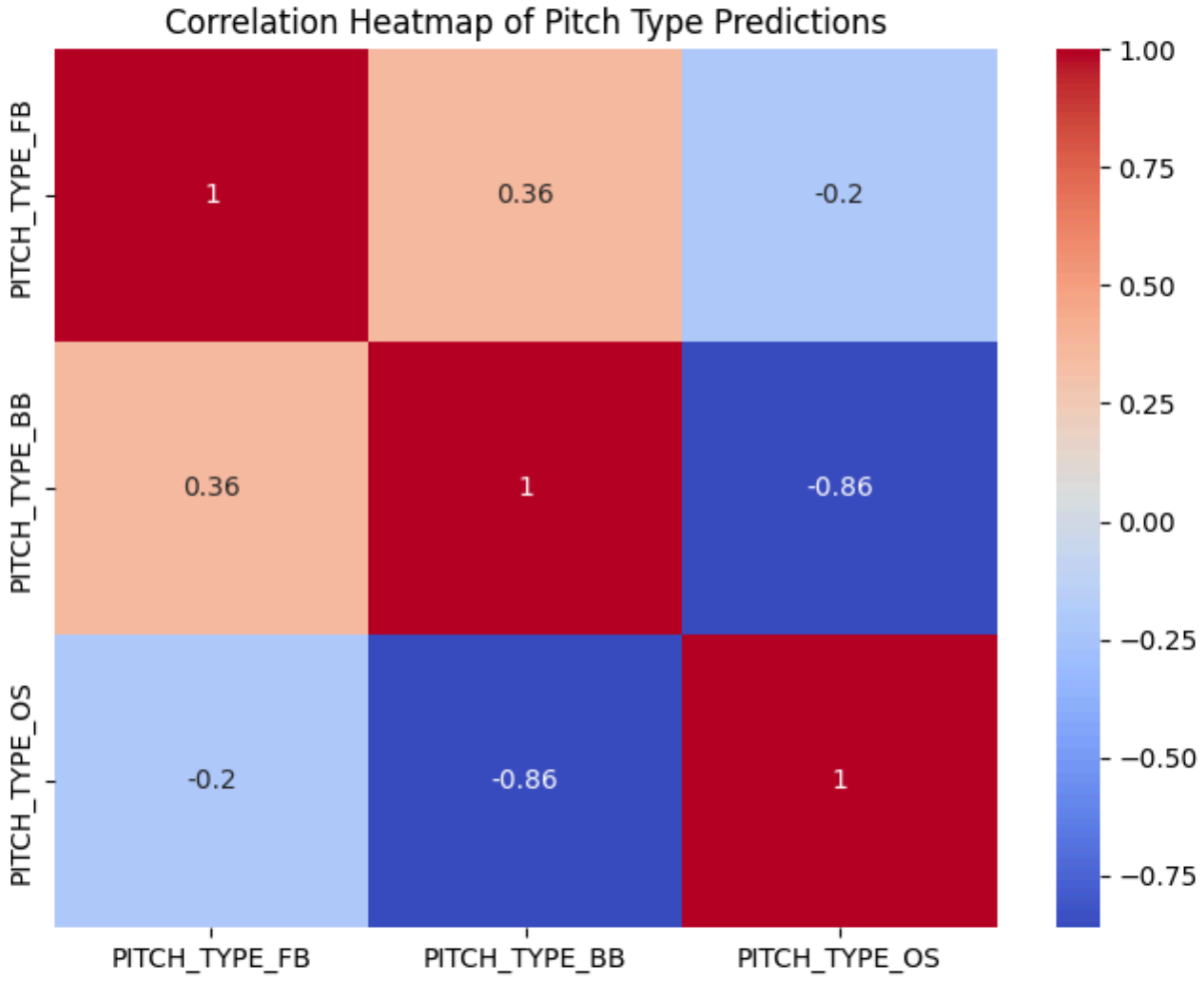
```
# 3. Scatter Plot (Fastball vs. Breaking Ball)
plt.figure(figsize=(8, 5))
sns.scatterplot(x='PITCH_TYPE_FB', y='PITCH_TYPE_BB', data=filtered_data)
plt.title('Fastball vs Breaking Ball Predictions')
plt.xlabel('PITCH_TYPE_FB')
plt.ylabel('PITCH_TYPE_BB')
plt.show()
```

```
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\laksh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: i
s_categorical_dtype is deprecated and will be removed in a future version. Use isinst
ance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [121…

```
# 4. Heatmap of Correlations
correlation_matrix = filtered_data[['PITCH_TYPE_FB', 'PITCH_TYPE_BB', 'PITCH_TYPE_OS'
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Pitch Type Predictions')
plt.show()
```



Conclusion

The pitch type prediction model provides valuable insights that can significantly impact game strategy for the Cincinnati Reds. By leveraging historical data and advanced modeling techniques, the team can make informed decisions to improve performance on the field.

End of Analysis