FIP (Fielding Independent Pitching) and WHIP (Walks plus Hits per Inning Pitched) are two important statistics used to evaluate the performance of pitchers in baseball. They provide insights into a pitcher's effectiveness and their ability to control the game.

1. \*\*FIP (Fielding Independent Pitching):\*\*

FIP is a statistic that aims to measure a pitcher's performance by focusing solely on aspects that a pitcher has control over – strikeouts, walks, hit-by-pitches, and home runs. It eliminates factors that are influenced by fielding defense, such as the number of hits allowed, which can vary based on the quality of the defense behind the pitcher.

The formula for FIP is:

\[ FIP = \frac{{(13 \times HR) + (3 \times BB) - (2 \times K)}}{{IP}} + FIP\\_constant \]

Where:

- \( HR \) is the number of home runs allowed.

- \( BB \) is the number of walks allowed.

- \( K \) is the number of strikeouts.

- \( IP \) is the number of innings pitched.

- \( FIP\\_constant \) is a constant value to scale FIP to look more like a traditional ERA.

FIP is important because it provides a clearer picture of a pitcher's true performance, regardless of the defense playing behind them. It helps identify pitchers who might have been unlucky due to poor defensive support and pitchers who consistently induce strikeouts and limit walks, which are generally more favorable outcomes for a pitcher.

2. \*\*WHIP (Walks plus Hits per Inning Pitched):\*\*

WHIP is a simple statistic that measures a pitcher's ability to prevent baserunners by calculating the average number of walks and hits allowed per inning pitched. A lower WHIP indicates better control and the ability to limit the number of runners on base.

The formula for WHIP is:

\[ WHIP = \frac{{BB + H}}{{IP}} \]

Where:

- \( BB \) is the number of walks allowed.

- \( H \) is the number of hits allowed.

- \( IP \) is the number of innings pitched.

WHIP is important because it provides a quick overview of a pitcher's ability to keep opponents off base. It's a good indicator of a pitcher's overall control, command, and effectiveness at preventing scoring opportunities.

In summary, FIP and WHIP are important pitcher statistics that help evaluate a pitcher's performance beyond traditional stats like ERA. FIP focuses on a pitcher's control over key aspects of their performance, while WHIP measures their ability to prevent baserunners. Both metrics provide valuable insights into a pitcher's skill set and overall effectiveness on the mound.

Certainly! ERA stands for Earned Run Average, and it's one of the most commonly used statistics to measure a pitcher's performance in baseball. ERA reflects how many earned runs a pitcher allows on average over nine innings and is an important indicator of a pitcher's ability to prevent opponents from scoring.

The formula for calculating ERA is straightforward:

\[ ERA = \frac{{\text{Earned Runs}} \times 9}}{{\text{Innings Pitched}}}\]

Where:

- "Earned Runs" are the runs that are scored by the opposing team due to hits, walks, and other offensive actions, excluding errors made by the fielding team.

- "Innings Pitched" is the total number of innings a pitcher has pitched.

ERA is essential because it provides a direct measure of a pitcher's effectiveness in preventing runs from scoring. A lower ERA indicates that a pitcher is more successful at keeping opponents from crossing home plate, which is a primary goal for any pitcher.

However, ERA has its limitations. It doesn't take into account factors like the quality of the defense playing behind the pitcher, luck in terms of where batted balls land, and the influence of ballpark dimensions. For this reason, advanced metrics like FIP, mentioned in the previous response, were developed to provide a more accurate assessment of a pitcher's individual performance, independent of external variables.

In summary, ERA is a widely recognized and used statistic to evaluate a pitcher's ability to prevent earned runs. It's a fundamental measure of a pitcher's performance, but it's important to consider other advanced metrics like FIP to gain a more comprehensive understanding of their effectiveness.

ETL:

All data for MLB starting pitchers was downloaded from FanGraphs for seasons 2018 – 2023.

1. **Data Cleaning and Manipulation**:
   * Filtered out rows with the season 2020 due to the pandemic-shortened year.
   * Null values in the DataFrame are filled with zeros.
   * Columns containing redundant stats were dropped from the DataFrame.
   * The data was filtered for specific relevant seasons (2018, 2019, 2021, 2022).
   * Data for the year 2023 is grouped by player ID.
   * Average statistics for ERA, FIP, and WHIP are calculated.
   * The average stats were merged with the 2023 stats for each player.
   * The pitcher names were merged with the results DataFrame.
2. **Z-Score Calculation**:
   * Z-scores were calculated for ERA, FIP, and WHIP for both the average values and the values for the year 2023.
   * The differences between z-scores were computed.
3. **Adding Z-Scores and Differences**:
   * The calculated z-scores and differences were added as new columns to the DataFrame.
4. **Output and Saving Data**:
   * The script outputs the merged DataFrame with z-scores and differences.
   * Intermediate values, standard deviations, and some specific columns are displayed.
5. **Creating Learning Datasets**:
   * Created separate datasets for learning ERA, FIP, and WHIP.
   * Specific columns were dropped from each dataset.
   * The modified datasets were saved as separate CSV files.

Analysis and ML:

Classification Results:

Logistic regression and XGBoost was used in training, evaluating, and visualizing the results of classification models. The goal was to predict performance metrics (ERA, FIP, WHIP) for pitchers.

1. \*\*Data Preprocessing:\*\*

- Imported necessary libraries and reading data from CSV files into separate DataFrames for ERA, FIP, and WHIP.

- Missing values in these DataFrames were imputed using the median strategy.

2. \*\*Logistic Regression Model for ERA, FIP, and WHIP:\*\*

- The `train\_and\_evaluate\_logistic\_regression` function used for a logistic regression model for (ERA, FIP, WHIP).

- The target columns were transformed into binary classes (0 or 1) based on whether the value is greater than 0.

- The data was split into training and testing sets, followed by applying SMOTE for oversampling to address class imbalances.

- Standardization using StandardScaler was applied to the features.

- Logistic regression model was trained, predictions were made, and accuracy calculated.

- A SHAP explainer was created to analyze feature importance.

3. \*\*Classification Model with XGBoost for ERA, FIP, and WHIP:\*\*

- The `train\_and\_evaluate\_classification\_model` function was evaluated using XGBoost classifier for each specific target column.

- Missing columns in the DataFrame were dropped, and the target column transformed similarly to the logistic regression case.

- Data was split, scaled, and a pipeline with SMOTE and XGBoost was created.

- Cross-validation scores were computed, the pipeline trained, and predictions made.

- SHAP values were calculated for feature importance.

4. \*\*Creating Predictions DataFrame:\*\*

- A function `create\_predictions\_dataframe` was defined to create a DataFrame containing predictions, actual values, and correctness (whether the prediction matches the actual value).

5. \*\*Running the Models for ERA, FIP, and WHIP:\*\*

- The code ran the logistic regression and classification models for ERA, FIP, and WHIP separately.

- The results, included accuracy and cross-validation scores, and were printed for each model.

- SHAP summary plots were generated to visualize feature importance.

6. \*\*Merging and Organizing Results:\*\*

- The prediction correctness data from ERA, FIP, and WHIP models were merged based on pitcher names.

- A DataFrame named `pitching\_verdict\_df` was created containing pitcher names and their correctness for ERA, FIP, and WHIP predictions.

7. \*\*Final Verdict DataFrame:\*\*

- The `pitching\_verdict\_df` DataFrame provides a consolidated view of correctness for ERA, FIP, and WHIP predictions for each pitcher.

Overall, the code performed classification tasks using logistic regression and XGBoost to predict the performance metrics of pitchers. It emphasizes accuracy, cross-validation scores, and SHAP values for feature importance analysis. The final DataFrame `pitching\_verdict\_df` summarizes the correctness of predictions for each performance metric and pitcher. The results can be used to assess the models' performance and provide insights into which features are most influential in making accurate predictions.

ERA Classification scores:

Cross Value Scores: [0.94814815 0.93333333 0.94814815 0.95522388 0.99253731]

Accuracy: 0.93

Recall: 0.93

ERA Shap graph

A screen shot of a graph

Description automatically generated

In the ERA prediction model, we a 5-fold cross-validation, with each of the trained subsets returning accuracy scores over 93%. The test data, or ratio of correct predictions to the total number of predictions made by the model was 93%. And finally, the true positive rate or sensitivity was 93%

Feature impact on ERA prediction (SHAP graph): The model suggests that higher values of WHIP\_avg, ERA\_avg, and FIP\_avg are strong contributors to higher predicted ERAs. And obviously, TG (Total Games) will account for higher predicted ERA’s with higher game appearances.

FIP training and evaluation results

Cross Value Scores: [0.93333333 0.94074074 0.96296296 0.95522388 0.97761194]

Accuracy: 0.95

Recall: 0.95

FIP SHAP Graph

A graph of different colored shapes

Description automatically generated

As we see above, the FIP (Fielding Independent Pitching) training model scored high. Unlike the old tried and true ERA stat, FIP considers only the factors that a pitcher can control (strikeouts, walks, home runs, etc…) with a pretty high degree of accuracy and recall.

One very interesting result in the SHAP for FIP, was that Pace moved up two spots in feature importance. Pace is a measure of the seconds between pitches for both hitters and pitchers based on PITCHf/x timestamps. However, as far as importance, most followers of Sabremetrics find Pace to have little impact on pitching performance, but the model seems to think it’s more important than we think.

WHIP training and evaluation scores:

Cross Value Scores: [0.92592593 0.95555556 0.96296296 0.94029851 0.97761194]

Accuracy: 0.92

Recall: 0.89

WHIP SHAP Graph

A graph of different colored shapes

Description automatically generated

WHIP (Walks and Hits per Inning Pitched) we continue to see that the 5-fold validation is returning some very good results. The held-out test data was above 90%, and good recall at 89%.

The SHAP graph for our WHIP prediction shows that the top features impacting our models appear to be pretty consistent for each targets.

CONCLUSIONS:

Overall, the models seemed to work very well. As you’ll see in the MLB Starting Pitcher Dashboard, the model predicted very closely to this year’s stats even though we’re not yet in the post season, with plenty of baseball still left to be analyzed. But I’d rather just watch the game.

CLASSIFICATION and XGBoost code:

**import** numpy **as** np

**import** pandas **as** pd

**import** xgboost **as** xgb

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**from** sklearn.metrics **import** mean\_squared\_error, accuracy\_score, recall\_score, classification\_report

**from** imblearn.over\_sampling **import** RandomOverSampler

**from** sklearn.linear\_model **import** LogisticRegression

**from** imblearn.pipeline **import** Pipeline, make\_pipeline

**from** imblearn.over\_sampling **import** SMOTE

**from** sklearn.model\_selection **import** train\_test\_split, cross\_val\_score, KFold

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.metrics **import** balanced\_accuracy\_score

**import** shap

In [2]:

pitcher\_full\_df **=** pd**.**read\_csv("Resources/full\_pitcher\_data.csv")

pitcher\_era\_df **=** pd**.**read\_csv("Resources/full\_era\_learning.csv")

pitcher\_fip\_df **=** pd**.**read\_csv("Resources/full\_fip\_learning.csv")

pitcher\_whip\_df **=** pd**.**read\_csv("Resources/full\_whip\_learning.csv")

In [3]:

*# Import the necessary libraries*

**from** sklearn.impute **import** SimpleImputer

*# Define the imputer with the strategy you want to use (median, mean, etc.)*

imputer **=** SimpleImputer(strategy**=**'median')

*# Columns with NaN values in each DataFrame*

columns\_with\_nans\_era **=** pitcher\_era\_df**.**columns[pitcher\_era\_df**.**isna()**.**any()]**.**tolist()

columns\_with\_nans\_fip **=** pitcher\_fip\_df**.**columns[pitcher\_fip\_df**.**isna()**.**any()]**.**tolist()

columns\_with\_nans\_whip **=** pitcher\_whip\_df**.**columns[pitcher\_whip\_df**.**isna()**.**any()]**.**tolist()

*# Apply the imputation to each DataFrame*

pitcher\_era\_df[columns\_with\_nans\_era] **=** imputer**.**fit\_transform(pitcher\_era\_df[columns\_with\_nans\_era])

pitcher\_fip\_df[columns\_with\_nans\_fip] **=** imputer**.**fit\_transform(pitcher\_fip\_df[columns\_with\_nans\_fip])

pitcher\_whip\_df[columns\_with\_nans\_whip] **=** imputer**.**fit\_transform(pitcher\_whip\_df[columns\_with\_nans\_whip])

In [4]:

pitcher\_era\_df **=** pitcher\_era\_df**.**drop(['Season','Team','Age Rng','z\_score\_era\_avg', 'z\_score\_fip\_avg', 'z\_score\_whip\_avg', 'z\_score\_era\_2023', 'z\_score\_fip\_2023','z\_score\_whip\_2023'], axis**=**1)

pitcher\_fip\_df **=** pitcher\_fip\_df**.**drop(['Season','Team','Age Rng','z\_score\_era\_avg', 'z\_score\_fip\_avg', 'z\_score\_whip\_avg', 'z\_score\_era\_2023', 'z\_score\_fip\_2023','z\_score\_whip\_2023'], axis**=**1)

pitcher\_whip\_df **=** pitcher\_whip\_df**.**drop(['Season','Team','Age Rng','z\_score\_era\_avg', 'z\_score\_fip\_avg', 'z\_score\_whip\_avg', 'z\_score\_era\_2023', 'z\_score\_fip\_2023','z\_score\_whip\_2023'], axis**=**1)

In [5]:

**def** train\_and\_evaluate\_logistic\_regression(df, target\_column, solver**=**'lbfgs', max\_iter**=**100):

*# Convert the target column into binary classes (0 or 1)*

df[target\_column] **=** df[target\_column]**.**apply(**lambda** x: 1 **if** x **>** 0 **else** 0)

*# Split data into features (X) and target (y)*

X **=** df**.**drop(target\_column, axis**=**1)

y **=** df[target\_column]

*# Split data into training and testing sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)

*# Apply SMOTE for oversampling*

smote **=** SMOTE(random\_state**=**42)

X\_train\_resampled, y\_train\_resampled **=** smote**.**fit\_resample(X\_train, y\_train)

*# Standardize features using StandardScaler*

scaler **=** StandardScaler()

X\_train\_scaled **=** scaler**.**fit\_transform(X\_train\_resampled)

X\_test\_scaled **=** scaler**.**transform(X\_test)

*# Create the Logistic Regression model*

logreg\_model **=** LogisticRegression(solver**=**solver, max\_iter**=**max\_iter)

*# Train the model*

logreg\_model**.**fit(X\_train\_scaled, y\_train\_resampled)

*# Make predictions*

y\_pred **=** logreg\_model**.**predict(X\_test\_scaled)

*# Calculate accuracy*

accuracy **=** accuracy\_score(y\_test, y\_pred)

*# Generate classification report*

class\_report **=** classification\_report(y\_test, y\_pred)

*# Create the SHAP explainer with the trained model*

explainer **=** shap**.**Explainer(logreg\_model, X\_train\_scaled)

**return** {

"accuracy": accuracy,

"classification\_report": class\_report,

"shap\_values": explainer(X\_test\_scaled),

"X\_test\_scaled": X\_test\_scaled,

"X\_train": X\_train\_resampled,

"X\_train\_scaled": X\_train\_scaled

}

In [6]:

pitcher\_era\_df

Out[6]:

|  | **Age** | **W** | **L** | **ERA** | **G** | **GS** | **CG** | **ShO** | **SV** | **BS** | **...** | **wNetPitV** | **TG** | **wOBA** | **OBP** | **SLG** | **wSB** | **ERA\_avg** | **FIP\_avg** | **WHIP\_avg** | **z\_score\_diff\_era** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 39 | 1 | 12 | 5.530121 | 22 | 22 | 0 | 0 | 0 | 0 | ... | 0.221085 | 132 | 0.0 | 0.0 | 0.0 | 0.0 | 3.381562 | 3.780352 | 1.121997 | 0.553782 |
| **1** | 35 | 11 | 4 | 2.515528 | 19 | 19 | 0 | 0 | 0 | 0 | ... | -0.263970 | 129 | 0.0 | 0.0 | 0.0 | 0.0 | 2.902238 | 3.155674 | 1.012831 | -0.120401 |
| **2** | 41 | 3 | 9 | 8.612070 | 17 | 17 | 0 | 0 | 0 | 0 | ... | -0.299898 | 131 | 0.0 | 0.0 | 0.0 | 0.0 | 3.854966 | 3.989132 | 1.307504 | 1.247842 |
| **3** | 37 | 2 | 6 | 6.264000 | 9 | 9 | 0 | 0 | 0 | 0 | ... | 0.000000 | 131 | 0.0 | 0.0 | 0.0 | 0.0 | 4.213119 | 3.648284 | 1.298954 | 0.510974 |
| **4** | 36 | 6 | 9 | 6.467967 | 21 | 21 | 0 | 0 | 0 | 0 | ... | 0.262619 | 131 | 0.0 | 0.0 | 0.0 | 0.0 | 4.011502 | 3.463603 | 1.283427 | 0.624339 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **837** | 24 | 0 | 1 | 135.000141 | 1 | 1 | 0 | 0 | 0 | 0 | ... | 0.000000 | 162 | 0.0 | 0.0 | 0.0 | 0.0 | 4.271903 | 4.204399 | 1.308840 | 0.041355 |
| **838** | 23 | 0 | 1 | 5.400000 | 1 | 1 | 0 | 0 | 0 | 0 | ... | 0.000000 | 162 | 0.0 | 0.0 | 0.0 | 0.0 | 4.271903 | 4.204399 | 1.308840 | 0.041355 |
| **839** | 24 | 0 | 0 | 11.117641 | 2 | 2 | 0 | 0 | 0 | 0 | ... | 0.000000 | 162 | 0.0 | 0.0 | 0.0 | 0.0 | 4.271903 | 4.204399 | 1.308840 | 0.041355 |
| **840** | 25 | 1 | 1 | 3.441176 | 7 | 7 | 0 | 0 | 0 | 0 | ... | 0.000000 | 161 | 0.0 | 0.0 | 0.0 | 0.0 | 4.271903 | 4.204399 | 1.308840 | 0.041355 |
| **841** | 25 | 0 | 0 | 0.000000 | 1 | 1 | 0 | 0 | 0 | 0 | ... | 0.000000 | 162 | 0.0 | 0.0 | 0.0 | 0.0 | 4.271903 | 4.204399 | 1.308840 | 0.041355 |

842 rows × 340 columns

In [7]:

target\_column **=** 'z\_score\_diff\_era'

result\_era **=** train\_and\_evaluate\_logistic\_regression(pitcher\_era\_df, target\_column, solver**=**'lbfgs', max\_iter**=**200)

*# Calculate Accuracy*

*#rmse = result\_era['rmse']*

accuracy **=** result\_era['accuracy']

shap\_values **=** result\_era['shap\_values']

print(f"Accuracy: {result\_era['accuracy']:.2f}")

Accuracy: 0.89

In [8]:

target\_column **=** 'z\_score\_diff\_fip'

result\_fip **=** train\_and\_evaluate\_logistic\_regression(pitcher\_fip\_df, target\_column, solver**=**'lbfgs', max\_iter**=**200)

*# Access the results*

print(f"Accuracy: {result\_fip['accuracy']:.2f}")

Accuracy: 0.87

In [9]:

target\_column **=** 'z\_score\_diff\_whip'

result\_whip **=** train\_and\_evaluate\_logistic\_regression(pitcher\_whip\_df, target\_column, solver**=**'lbfgs', max\_iter**=**200)

*# Access the results*

print(f"Accuracy: {result\_whip['accuracy']:.2f}")

Accuracy: 0.89

In [10]:

**def** train\_and\_evaluate\_classification\_model(df, target\_column):

*# Drop rows with missing values*

df**.**dropna(axis**=**1, inplace**=True**)

*# Convert the target column into binary classes (0 or 1)*

df[target\_column] **=** df[target\_column]**.**apply(**lambda** x: 1 **if** x **>** 0 **else** 0)

*# Split data into features (X) and target (y)*

X **=** df**.**drop(target\_column, axis**=**1)

y **=** df[target\_column]

*# Split data into training and testing sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**42)

*# Standardize features using StandardScaler*

scaler **=** StandardScaler()

X\_train\_scaled **=** scaler**.**fit\_transform(X\_train)

X\_test\_scaled **=** scaler**.**transform(X\_test)

*# Define cross-validation strategy*

kf **=** KFold(n\_splits**=**5, shuffle**=True**, random\_state**=**42)

*# Build the pipeline*

imb\_pipeline **=** Pipeline([

('smote', SMOTE(random\_state**=**42)),

('xgbclassifier', xgb**.**XGBClassifier(

max\_depth**=**5,

learning\_rate**=**0.01,

n\_estimators**=**600,

subsample**=**0.5,

colsample\_bytree**=**0.25,

objective**=**'binary:logistic',

random\_state**=**42

))

])

*# Cross-validate the pipeline*

cross\_val\_scores **=** cross\_val\_score(imb\_pipeline, X\_train\_scaled, y\_train, scoring**=**'accuracy', cv**=**kf)

*# Train the pipeline on the training data*

imb\_pipeline**.**fit(X\_train\_scaled, y\_train)

*# Make predictions on the test set*

y\_pred **=** imb\_pipeline**.**predict(X\_test\_scaled)

*# Create the SHAP explainer*

final\_estimator **=** imb\_pipeline**.**named\_steps['xgbclassifier']

explainer **=** shap**.**Explainer(final\_estimator, X\_train\_scaled)

*# Calculate SHAP values*

shap\_values **=** explainer(X\_test\_scaled, check\_additivity**=False**)

*# Calculate evaluation metrics*

balanced\_recall **=** balanced\_accuracy\_score(y\_test, y\_pred)

accuracy **=** accuracy\_score(y\_test, y\_pred)

**return** {

"cross\_val\_scores": cross\_val\_scores,

"accuracy": accuracy,

"balanced\_recall": balanced\_recall,

"shap\_values": shap\_values,

"X\_test\_scaled": X\_test\_scaled,

"X\_train": X\_train,

"X\_train\_scaled": X\_train\_scaled,

"y\_pred": y\_pred,

"y\_test": y\_test,

"y\_train": y\_train,

"X\_test": X\_test

}

In [11]:

**def** create\_predictions\_dataframe(result\_df, name\_df):

y\_pred\_train **=** result\_df['y\_train']

y\_pred\_test **=** result\_df['y\_pred']

y\_train **=** result\_df['y\_train']

y\_test **=** result\_df['y\_test']

indexes\_train **=** result\_df['X\_train']**.**index

indexes\_test **=** result\_df['X\_test']**.**index

names\_train **=** name\_df**.**loc[indexes\_train, 'Name']

predictions\_with\_names\_train **=** list(zip(names\_train, y\_pred\_train, y\_train))

names\_test **=** name\_df**.**loc[indexes\_test, 'Name']

predictions\_with\_names\_test **=** list(zip(names\_test, y\_pred\_test, y\_test))

predictions\_df\_train **=** pd**.**DataFrame(predictions\_with\_names\_train, columns**=**['Name', 'Prediction', 'Actual'])

predictions\_df\_test **=** pd**.**DataFrame(predictions\_with\_names\_test, columns**=**['Name', 'Prediction', 'Actual'])

*# Add a column for correctness*

predictions\_df\_train['Correctness'] **=** np**.**where(predictions\_df\_train['Prediction'] **==** predictions\_df\_train['Actual'], 'Correct :)', 'Wrong :(')

predictions\_df\_test['Correctness'] **=** np**.**where(predictions\_df\_test['Prediction'] **==** predictions\_df\_test['Actual'], 'Correct :)', 'Wrong :(')

pred\_actual\_df **=** pd**.**concat([predictions\_df\_train, predictions\_df\_test], ignore\_index**=True**)

**return** pred\_actual\_df

In [12]:

result\_era **=** train\_and\_evaluate\_classification\_model(pitcher\_era\_df, 'z\_score\_diff\_era')

*# Print the results*

print(f"Cross Value Scores: {result\_era ['cross\_val\_scores']}")

print(f"Accuracy: {result\_era ['accuracy']:.2f}")

print(f"Recall: {result\_era ['balanced\_recall']:.2f}")

Cross Value Scores: [0.94814815 0.93333333 0.94814815 0.95522388 0.99253731]

Accuracy: 0.93

Recall: 0.93

In [13]:

*# Generate SHAP summary plot*

plt**.**figure(figsize**=**(8, 6))

shap**.**summary\_plot(result\_era['shap\_values'], result\_era['X\_test\_scaled'], plot\_type**=**'violin', feature\_names**=**result\_era['X\_train']**.**columns, show**=False**)

ax **=** plt**.**gca()

ax**.**figure**.**set\_size\_inches(8, 6)

plt**.**savefig('Images/shap\_era\_class.png', bbox\_inches**=**'tight')

plt**.**tight\_layout()

plt**.**show()

No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored

A graph of different colored lines

Description automatically generated

In [14]:

pred\_actual\_era **=** create\_predictions\_dataframe(result\_era, pitcher\_full\_df)

new\_columns **=** {

"Prediction": "ERA\_Prediction",

"Actual": "ERA\_Actual",

"Correctness": "ERA\_Correctness"

}

pred\_actual\_era**=** pred\_actual\_era**.**rename(columns**=**new\_columns)

In [15]:

result\_fip **=** train\_and\_evaluate\_classification\_model(pitcher\_fip\_df, 'z\_score\_diff\_fip')

*# Print the results*

print(f"Cross Value Scores: {result\_fip ['cross\_val\_scores']}")

print(f"Accuracy: {result\_fip ['accuracy']:.2f}")

print(f"Recall: {result\_fip ['balanced\_recall']:.2f}")

Cross Value Scores: [0.93333333 0.94074074 0.96296296 0.95522388 0.97761194]

Accuracy: 0.95

Recall: 0.95

In [16]:

*# Generate SHAP summary plot*

plt**.**figure(figsize**=**(8, 6))

shap**.**summary\_plot(result\_fip['shap\_values'], result\_fip['X\_test\_scaled'], plot\_type**=**'violin', feature\_names**=**result\_fip['X\_train']**.**columns, show**=False**)

ax **=** plt**.**gca()

ax**.**figure**.**set\_size\_inches(8, 6)

plt**.**savefig('Images/shap\_fip\_class.png', bbox\_inches**=**'tight')

plt**.**tight\_layout()

plt**.**show()

No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored

A graph of different colored shapes

Description automatically generated with medium confidence

In [17]:

pred\_actual\_fip **=** create\_predictions\_dataframe(result\_fip, pitcher\_full\_df)

new\_columns **=** {

"Prediction": "FIP\_Prediction",

"Actual": "FIP\_Actual",

"Correctness": "FIP\_Correctness"

}

pred\_actual\_fip**=** pred\_actual\_fip**.**rename(columns**=**new\_columns)

In [18]:

result\_whip **=** train\_and\_evaluate\_classification\_model(pitcher\_whip\_df, 'z\_score\_diff\_whip')

*# Print the results*

print(f"Cross Value Scores: {result\_whip ['cross\_val\_scores']}")

print(f"Accuracy: {result\_whip ['accuracy']:.2f}")

print(f"Recall: {result\_whip ['balanced\_recall']:.2f}")

Cross Value Scores: [0.92592593 0.95555556 0.96296296 0.94029851 0.97761194]

Accuracy: 0.92

Recall: 0.89

In [19]:

*# Generate SHAP summary plot*

plt**.**figure(figsize**=**(8, 6))

shap**.**summary\_plot(result\_whip['shap\_values'], result\_whip['X\_test\_scaled'], plot\_type**=**'violin', feature\_names**=**result\_whip['X\_train']**.**columns, show**=False**)

ax **=** plt**.**gca()

ax**.**figure**.**set\_size\_inches(8, 6)

plt**.**savefig('Images/shap\_whip\_class.png', bbox\_inches**=**'tight')

plt**.**tight\_layout()

plt**.**show()

No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored

A graph of different colored shapes

Description automatically generated

In [20]:

pred\_actual\_whip **=** create\_predictions\_dataframe(result\_whip, pitcher\_full\_df)

new\_columns **=** {

"Prediction": "WHIP\_Prediction",

"Actual": "WHIP\_Actual",

"Correctness": "WHIP\_Correctness"

}

pred\_actual\_whip**=** pred\_actual\_whip**.**rename(columns**=**new\_columns)

In [21]:

*# Merge the DataFrames on the "Name" column*

merged\_df **=** pd**.**merge(pred\_actual\_era, pred\_actual\_fip, on**=**"Name")

merged\_df **=** pd**.**merge(merged\_df, pred\_actual\_whip, on**=**"Name")

pitching\_verdict\_df **=** merged\_df**.**drop(["ERA\_Prediction", "ERA\_Actual", "FIP\_Prediction", "FIP\_Actual",

"WHIP\_Prediction", "WHIP\_Actual"], axis**=**1)

pitching\_verdict\_df**.**head()

Out[21]:

|  | **Name** | **ERA\_Correctness** | **FIP\_Correctness** | **WHIP\_Correctness** |
| --- | --- | --- | --- | --- |
| **0** | Jonathan Heasley | Correct :) | Correct :) | Correct :) |
| **1** | Chris Volstad | Correct :) | Correct :) | Correct :) |
| **2** | Robert Gsellman | Correct :) | Correct :) | Correct :) |
| **3** | Angel Zerpa | Correct :) | Correct :) | Correct :) |
| **4** | Aníbal Sánchez | Correct :) | Correct :) | Correct :) |

In [23]:

pitching\_verdict\_df**.**to\_csv('Resources/pitching\_verdict.csv', index**=False**)