Developing A MLB Swing/Miss Model With Random Forests

1. Introduction

With the rise in the adoption of analytics throughout the MLB, there have been a rise in occurrences of certain strategies. One of these offensive strategies is the increase in hitters who have "Three True Outcomes". This saying pertains to hitters who hit a lot of home runs, strikeout a lot, and walk a lot. The hitters that follow this strategy often have a high swing and miss rate causing the higher amount of strikeouts. Despite the increase in this offensive approach, due to the new rule changes inacted during the 2023 MLB season, a new offensive strategy may be more effective. Pitch clock rules have already shown to lead to a higher rate in stolen bases while the "No-Shift" rule will most likely lead to more hits. The reason these changes may lead to an overall strategy change is that it is easier to get on base and then advance. This diminishes the need to hit for power as there are alternative solutions to reach a threating position to score. With a potential rise in the importance of contact, we will need a better way to model a batter's ability to put balls in play as well as a pitcher's ability to generate swings and misses.

This brings us to the context and goal of this project. The goal of this research is to fit a model in order to predict swing and misses depending on the metrics of the pitch like velocity, effective velocity, and much more that will be discussed in the following section. The purpose behind this is to fit an effective model and then measure hitters on their true swing and miss rate compared with their expected swing and miss rate from the model along with measure pitchers and the expected swings and misses their pitches generate. This will accomplish two things, allow us to quantify hitters who are truly better at making contact based on the pitches they see and quantify pitchers whose pitches have a higher rate of expected swings and misses. With the new rule changes and even if the rules stayed constant, this type of model may hold significance in grading player effectiveness and productivity.

2. Data

The data used for this research was accumulated from the baseballR package in R with the statcast_search_pitchers() function. Due to taking every single pitch from the 2022 MLB season the data was rather large and I had to call this function for each day individually. The script to compile the data, clean it, and export it is included in the GitHub respository. Some cleaning was done in this notebook as well which can be seen in the code chunk. Below is a table containing all the columns included in the dataset and a brief description of each.

In [1]:
```
# Importing Data With CSV File
import pandas as pd
```

```
from tabulate import tabulate
pitches = pd.read_csv('C:/Users/tscot/Downloads/PersonalProjects/swingmissproj/pitches

# Creating New Columns For Handedniss
RR = []
LR = []
LL  = []
RL = []
for i in range(pitches.shape[0]):
    if pitches['stand'][i] == "R" and pitches['p_throws'][i] == "R":
        RR.append(1)
        LR.append(0)
        LL.append(0)
        RL.append(0)
    elif pitches['stand'][i] == "R" and pitches['p_throws'][i] == "L":
        RR.append(0)
        LR.append(0)
        LL.append(0)
        RL.append(1)
    elif pitches['stand'][i] == "L" and pitches['p_throws'][i] == "R":
        RR.append(0)
        LR.append(1)
        LL.append(0)
        RL.append(0)
    else:
        RR.append(0)
        LR.append(0)
        LL.append(1)
        RL.append(0)

# Removing uneeded columns and adding new columns
pitches = pitches.iloc[:,1:]
pitches['pitcher_name'] = pitches['player_name']
pitches = pitches.drop(['X','description', 'des', 'game_pk','stand', 'p_throws', 'play
added = {"RR":RR, "LR":LR, "LL":LL, "RL":RL}
pitches = pd.concat([pitches, pd.DataFrame(added)], axis = 1)

# Creating Description Table
desc = ['Type of pitch thrown', 'Velocity of pitch when released', 'Lateral spatial lo
       'Depth of ball release', 'Name of Pitcher', 'Horizontal movement of pitch',
       'Vertical movement of pitch', 'Horizontal position of ball crossing plate',
        'Vertical position of ball crossing plate', 'Velocity of pitch in X dimension'
       'Velocity of pitch in Y dimension', 'Velocity of pitch in Z dimension',
       'Acceleration of pitch in X dimension', 'Acceleration of pitch in Y dimension',
       'Acceleration of pitch in Z dimension', 'Top of strike zone', 'Bottom of strike
       'Effective speed of pitch', 'Spin rate at release', 'Extension of release', 'Ve
       'Spin axis of pitch', 'Name of Batter', 'Swing and Miss', "Right vs. Right (Bat
        "Left vs. Left", "Right vs. Left"]

description = {'Column Name': pitches.columns, 'Description': desc}
print(tabulate(pd.DataFrame(description), headers = ['Variable Name', 'Description'],
```

	Variable Name	Description
0	pitch_type	Type of pitch thrown
1	release_speed	Velocity of pitch when released
2	release_pos_x	Lateral spatial location of release
3	release_pos_z	Depth of ball release
4	pitcher_name	Name of Pitcher
5	pfx_x	Horizontal movement of pitch
6	pfx_z	Vertical movement of pitch
7	plate_x	Horizontal position of ball crossing plate
8	plate_z	Vertical position of ball crossing plate
9	vx0	Velocity of pitch in X dimension
10	vy0	Velocity of pitch in Y dimension
11	vz0	Velocity of pitch in Z dimension
12	ax	Acceleration of pitch in X dimension
13	ay	Acceleration of pitch in Y dimension
14	az	Acceleration of pitch in Z dimension
15	sz_top	Top of strike zone
16	sz_bot	Bottom of strike zone
17	effective_speed	Effective speed of pitch
18	release_spin_rate	Spin rate at release
19	release_extension	Extension of release
20	release_pos_y	Vertial spatial location of release
21	spin_axis	Spin axis of pitch
22	batter_name	Name of Batter
23	miss	Swing and Miss
24	RR	Right vs. Right (Batter First)
25	LR	Left vs. Right
26	LL	Left vs. Left
27	RL	Right vs. Left

Below is a portion of the final data set and various rows of some of the columns in order to give an idea of the type of values. Not all columns are included due to the size of the data itself however, the CSV file is uploaded within the repository and able to be used or examined.

In [2]:
```
# Head Of Data
print(pitches.iloc[0:45, [0,1,2,3,5,6]].head())
```

```
  pitch_type  release_speed  release_pos_x  release_pos_z  pfx_x  pfx_z
0         FF           95.3          -1.88           5.57  -0.33   1.51
1         FF           95.5          -1.82           5.67  -0.23   1.43
2         SI           88.5          -0.93           6.37  -0.74   0.94
3         CU           78.3           0.77           5.88  -0.77  -1.40
4         CH           84.1           1.71           5.71   1.14   0.04
```

3. Methodology

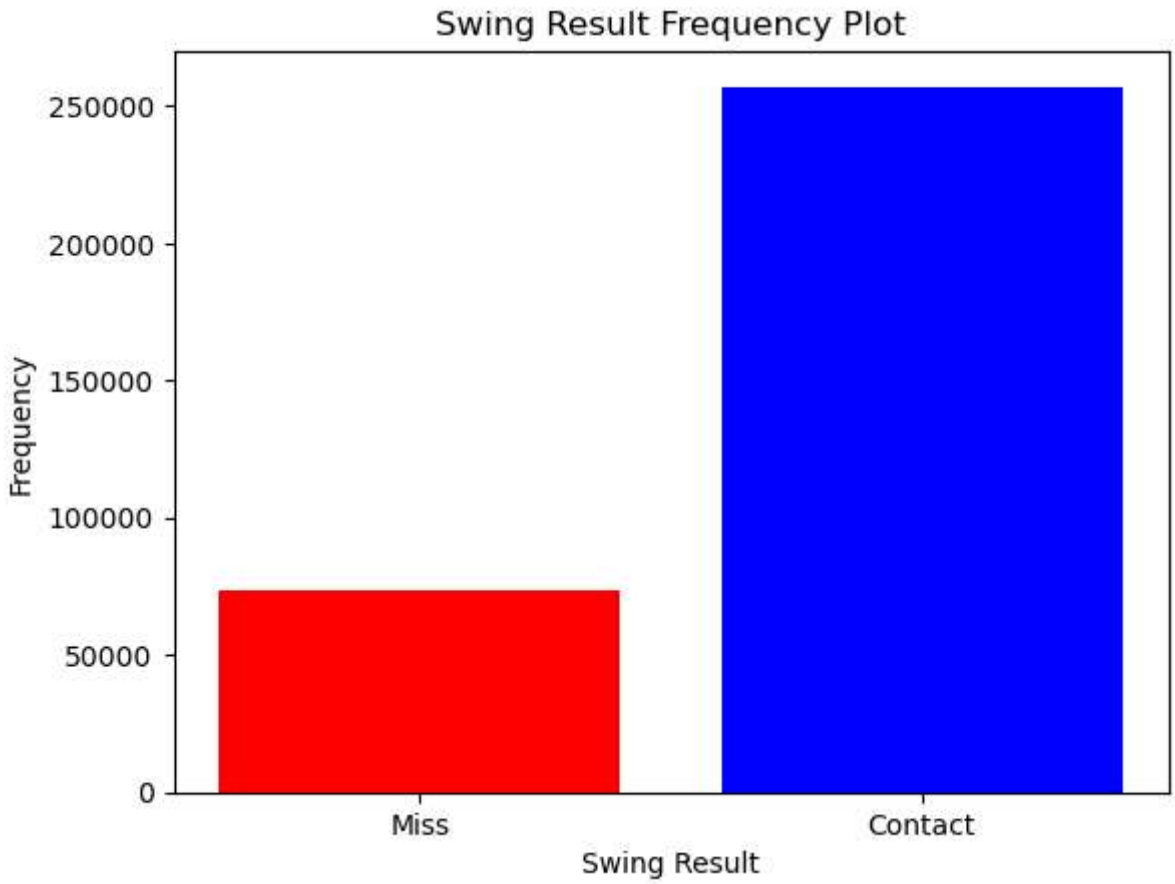
3.1. Algorithm

Due to the goals of the model and predicting whether a pitch is a swing results in contact or a miss, a classification algorithm will be employed. After utilizing multiple different algorithms in the research process, the random forest algorithm was selected. This was for a multitude of reasons however, the main reason was the need for an algorithm to perform effectively given weights assigned to each class. As seen in the below plot, swings were far more likely to result

in contact (foul or in play) rather than a swing and a miss. Due to this nature, the model could perform at high rates simply by predicting everything to be contact. While the accuracy would be high, this defeats the purpose of fitting such a model. To combat this, weights are needed to effectively penalize the algorithm more when a whiff is incorrectly classified. This allows the model to equally focus on both outcomes and create a more effective model in the context of this study. Random forests are able to include weighting of classification categories effectively and was a successful approach as seen later on in the paper.

As for the algorithm itself, a random forest is a collection of many different decision trees that all have different components. For this research, each model has 100 decision trees that make it up. When fitting the model, the top 100 performing decision trees are selected based off in-sample accuracy to make up the random forest. When used upon testing data, the classification prediction is formed based on which category the majority of the decision trees predict. This algorithm is more effective than using a single decision tree as it captures trends more effectively in the data with multiple trees rather than the one tree alone. Due to containing many different decision trees with even more different attributes, it is hard to display the model like how a linear regression equation would be. Instead of these coefficients we can see which predictors held the most significance throughout the forest and will be investigated later on during the model fitting stage of the research.

In [3]:
```
# Importing Module
import matplotlib.pyplot as plt
import numpy as np

# Training Data Counts
labels = ["Miss", "Contact"]
result = [sum(pitches['miss']), len(pitches['miss']) - sum(pitches['miss'])]

# Plotting Data
plt.bar(labels, result, color = ['red', 'blue'])

# Creating Labels And Legend
plt.xlabel('Swing Result')
plt.ylabel('Frequency')
plt.title('Swing Result Frequency Plot')

# Output
plt.show()
```



3.2. Model Components

As for the model itself, it will be a collection of multiple different models for each pitch type. Each pitch interacts differently in terms of spin rate and movement. Due to this, it would be inappropriate to try modeling all types of pitches in one model. By specifying the pitch, there should be a higher degree of acccurracy as well as gained insights. Not only will player effectiveness in terms of contact be represented but, the individual models can show players and which pitches have a higher impact on them. There will be four groups of pitches and four subsequent models. The four groups will be fastballs, offspeed, curveballs, and sliders. These groups are compiled of multiple similar types of pitches which will be shown in the table below.

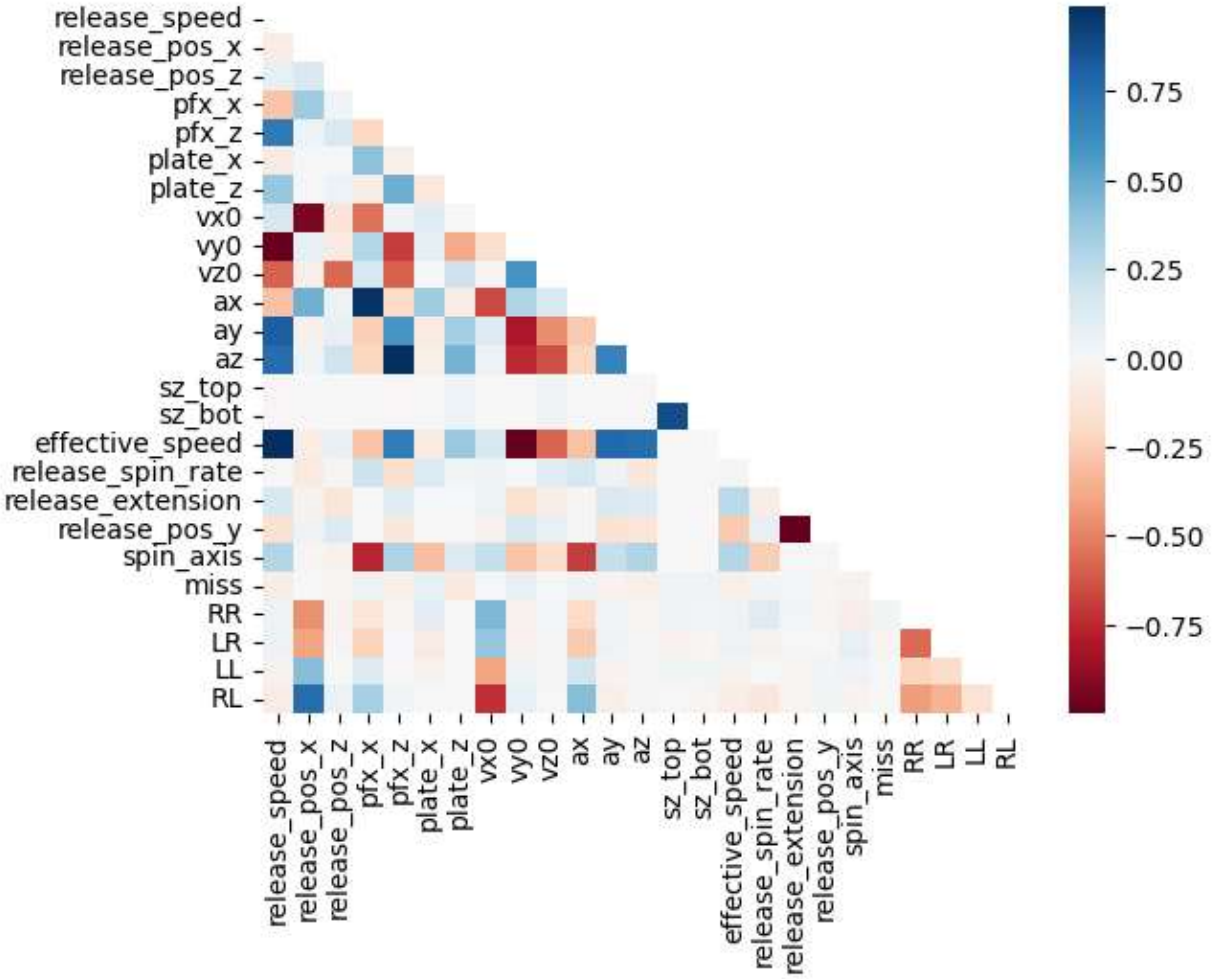
In [8]:
```
# Building Groups Table
pitchgroup = ['Fastball', 'Offspeed', 'Curveball', 'Slider']
groupings = ['4-Seam Fastball, Cutter, Sinker', 'Changeup, Split-Finger',
             'Curveball, Knuckle Curve, Slow Curve', 'Slider, Sweeper, Slurve']

group = {"Pitch Category": pitchgroup, "Pitches Included": groupings}

print(pd.DataFrame(group))
```

```
  Pitch Category                     Pitches Included
0       Fastball       4-Seam Fastball, Cutter, Sinker
1       Offspeed                 Changeup, Split-Finger
2      Curveball  Curveball, Knuckle Curve, Slow Curve
3         Slider               Slider, Sweeper, Slurve
```

4. Exploratory Data Analysis

Even though the random forest algorithm will essentially do all the work in creating our predictions, it is still insightful to investigate the data before the model fitting process to identify key trends or ideas in the data. It is good practice to have an idea how the model will react to the training data in order to identify potential errors. Right off the bat, using the heatmap, we can see that the variables 'vy0' and 'release_pos_y' are not needed because 'release_speed' and 'release extension' respectively already contain these attributes. Besides duplicates, it is interesting to see that most columns do not have strong correlations while some do. When investigating further, many of the correlated variables are due to their relation to each other. For example, the bottom of the strike zone is correlated with the top of the zone which makes sense as the two rely on each other. This can also be seen with pitch velocity and pitch acceleration as the two are both related to each other to some degree. Besides the expected correlations, there are a few interesting relationships between variables such as 'vx0' and the binary markers of the handidness of the matchup such as 'RL'. For all four matchups, 'vx0', the horizontal velocity, has somewhat of a significant relationship. While this is worth noting, it should be noted that these correlations may be due to the matchup variables being binary and therefore not a linear relationship. Another interesting group of relationships are the movements of the pitch with the velocities of the pitch. All in all, there seem to be some correlated variables while the majority do not, meaning they shold allow for a large capture of variation in the response variable, swings and misses.

In [4]:
```
import seaborn as sb

# HeatMap Visualization
corrmat = pitches.drop(['pitcher_name', 'batter_name'], axis = 1)
corr = corrmat.corr(numeric_only = True)
mask = np.triu(np.ones_like(corr, dtype = bool))
sb.heatmap(corr, mask = mask, cmap = "RdBu")

# Removing Repeated Columns
pitches = pitches.drop(['vy0', 'release_pos_y'], axis = 1)
```



5. Developing The Models

5.1 Prepping Data & Model Fitting

Next, the model will actually be fitted using the aforementioned random forest algorithm. The pitches dataset is divided up into the four pitch types with each model including weights based off the occurrences of a swing and a miss for each pitch type. What this means is that there won't be uniform weights across all four pitch models and each one will have its own depending of the occurrences of misses for that particular pitch. All random states were set to 1 for each model in order to assure replicability and as previously stated, each pitch type had its own model fitted. The output below is generated from fitting the data and doesn't hold any particular signifcance besides that the fitting process has been completed. The particular function used to fit the random forest algorithm was the 'RandomForestClassifier' function from 'sklearn.ensemble'. The feature importance of the fitted models will be discussed in the next subsection.

In [5]:
```
# Seperating The 4 Different Data Sets
## Fastball Model Data
fast_data = pitches[pitches['pitch_type'].isin(['FF', 'FC', 'SI'])]
fast_miss = fast_data['miss']
fast_data = fast_data.drop(['miss', 'pitch_type'], axis = 1)
```

```
fast_exp_2022 = fast_data
fast_data = fast_data.drop(['pitcher_name', 'batter_name'], axis = 1)

## Offspeed Model Data
off_data = pitches[pitches['pitch_type'].isin(['CH', 'FS'])]
off_miss = off_data['miss']
off_data = off_data.drop(['miss', 'pitch_type'], axis = 1)
off_exp_2022 = off_data
off_data = off_data.drop(['pitcher_name', 'batter_name'], axis = 1)

## Curveball Model Data
curve_data = pitches[pitches['pitch_type'].isin(['KC', 'CU', 'CS'])]
curve_miss = curve_data['miss']
curve_data = curve_data.drop(['miss', 'pitch_type'], axis = 1)
curve_exp_2022 = curve_data
curve_data = curve_data.drop(['pitcher_name', 'batter_name'], axis = 1)

## Slider Model Data
slid_data = pitches[pitches['pitch_type'].isin(['SL', 'ST', 'SV'])]
slid_miss = slid_data['miss']
slid_data = slid_data.drop(['miss', 'pitch_type'], axis = 1)
slid_exp_2022 = slid_data
slid_data = slid_data.drop(['pitcher_name', 'batter_name'], axis = 1)

# Fitting Models
## Importing Needed Modules
from sklearn.model_selection import KFold
import numpy as np

## Calculating Weights
fast_miss_percentage = sum(fast_miss)/len(fast_miss)
cw_fast = {0:fast_miss_percentage, 1:1-fast_miss_percentage}


off_miss_percentage = sum(off_miss)/len(off_miss)
cw_off = {0:off_miss_percentage, 1:1-off_miss_percentage}

curve_miss_percentage = sum(curve_miss)/len(curve_miss)
cw_curve = {0:curve_miss_percentage, 1:1-curve_miss_percentage}

slid_miss_percentage = sum(slid_miss)/len(slid_miss)
cw_slid = {0:slid_miss_percentage, 1:1-slid_miss_percentage}

# Import Random Forest Module
from sklearn.ensemble import RandomForestClassifier

# Initializing Objects

fast_tree = RandomForestClassifier(class_weight = cw_fast, random_state = 1)
off_tree = RandomForestClassifier(class_weight = cw_off, random_state = 1)
curve_tree = RandomForestClassifier(class_weight = cw_curve, random_state = 1)
slid_tree = RandomForestClassifier(class_weight = cw_slid, random_state = 1)

# Fitting Models
fast_tree.fit(fast_data, fast_miss)
off_tree.fit(off_data, off_miss)
curve_tree.fit(curve_data, curve_miss)
slid_tree.fit(slid_data, slid_miss)
```

Out[5]:

```
RandomForestClassifier(class_weight={0: 0.2979194773840801,
                                     1: 0.7020805226159199},
                       random_state=1)
```

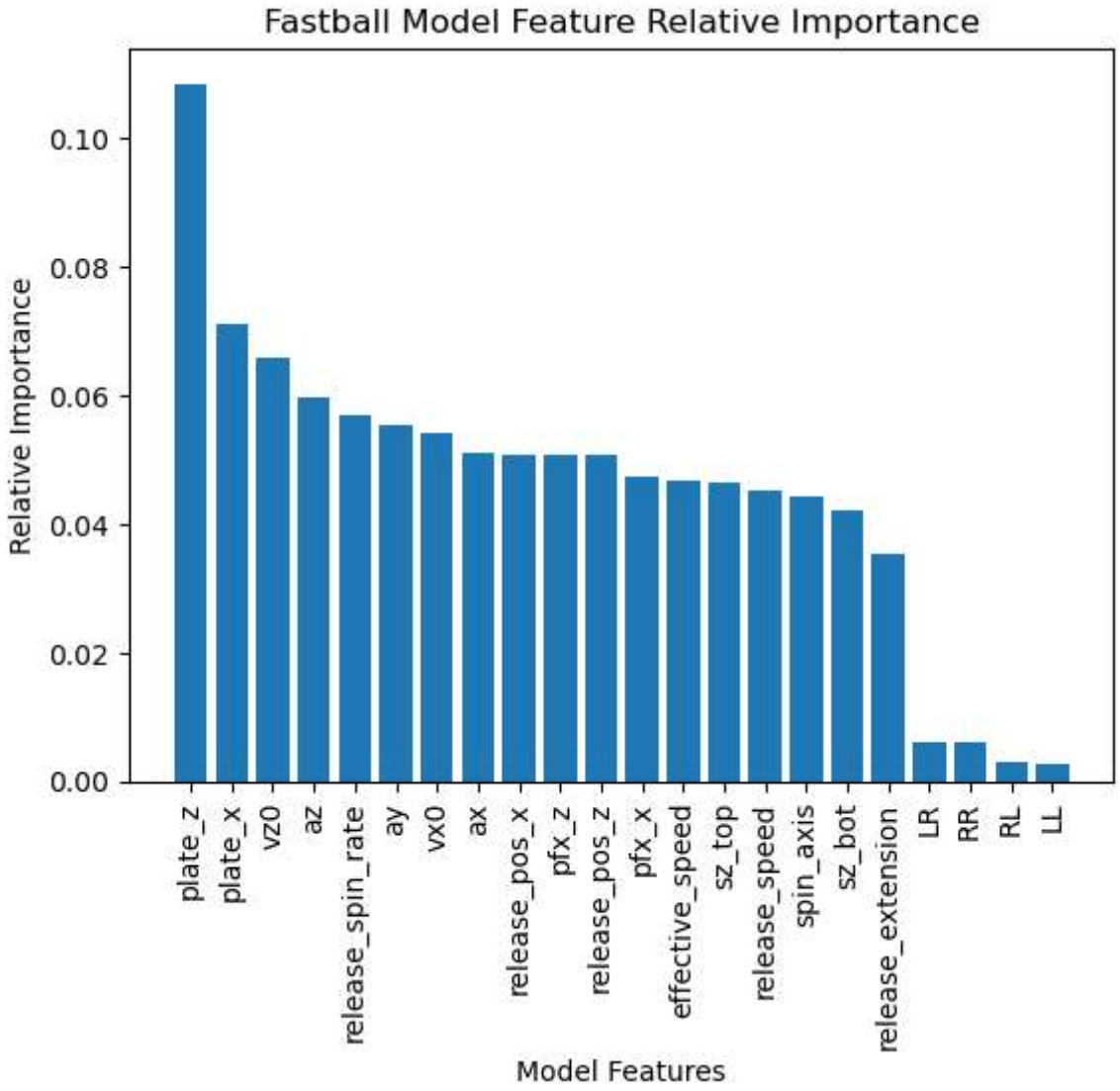
5.2. Model Feature Importance Plots

5.2.1. Fastball Model

Below is the relative feature importance plot for the fastball model. Looking at the results, location of the pitch is the most important feature of the pitch. Both the vertical location and horizontal location of the ball when it crosses the plate have the highest importance when determining whether it was a swing and a miss. This makes sense as a pitch that is nowhere near the plate is way more likely to be a miss while a pitch right down the middle is "easier" and more likely to be hit. The next most important feature, is "vz0" or the velocity of the pitch vertically. This feature is interesting as fastballs tend to not move as much as other pitches and are effective for their velocity. With this feature being so important it seems that fastball movement is more important when trying to get a swing and miss rather than velocity. The next most important feature is "az0" which is the vertical acceleration of the pitch. These two features are relatively the same as one is a function of the other and not suprising that they have similar feature importance. These features being so important seem to support the idea that sinkers are better for swings and misses than the traditional fastball as sinkers tend to have more vertical movement.

In [11]:

```
# Feature Importance Calls
fast_mod_imp = fast_tree.feature_importances_

# Fastball Model Feature Importance Plot
sort = np.argsort(fast_mod_imp)[::-1]
fast_s_imp = fast_mod_imp[sort]
fast_s_columns = [fast_data.columns[i] for i in sort]

## Plot Code
plt.bar(range(len(fast_s_imp)), fast_s_imp)
plt.xlabel('Model Features')
plt.ylabel('Relative Importance')
plt.title('Fastball Model Feature Relative Importance')
plt.xticks(range(len(fast_s_columns)), fast_s_columns, rotation = "vertical")
plt.show()
```

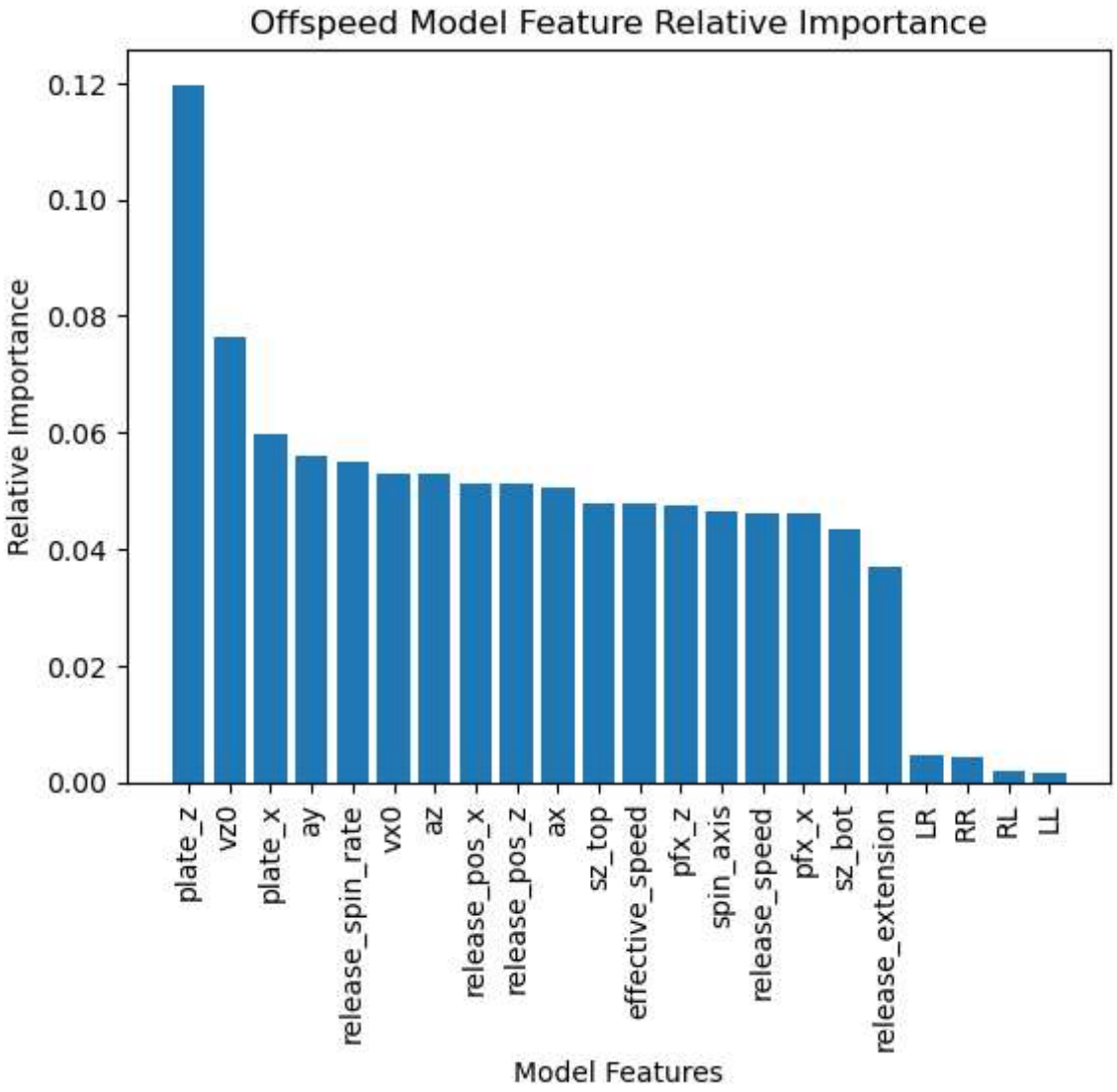


5.2.2. Offspeed Model

Similarly like the fastball model, the vertical location of the pitch has the highest relative importance for all the features in the model. Despite the horizontal location mattering the most when determing contact or a miss, it is not the second most important feature. The horizontal location is the third most important feature, while the vertical velocity is the second most important feature. Besides the first three features, the rest of the features in the model hold importance however they are all relatively the same.

In [12]:
```
off_mod_imp = off_tree.feature_importances_

# Offspeed Model Feature Importance Plot
sort = np.argsort(off_mod_imp)[::-1]
off_s_imp = off_mod_imp[sort]
off_s_columns = [off_data.columns[i] for i in sort]

## Plot Code
plt.bar(range(len(off_s_imp)), off_s_imp)
plt.xlabel('Model Features')
plt.ylabel('Relative Importance')
plt.title('Offspeed Model Feature Relative Importance')
```

```
plt.xticks(range(len(off_s_columns)), off_s_columns, rotation = "vertical")
plt.show()
```

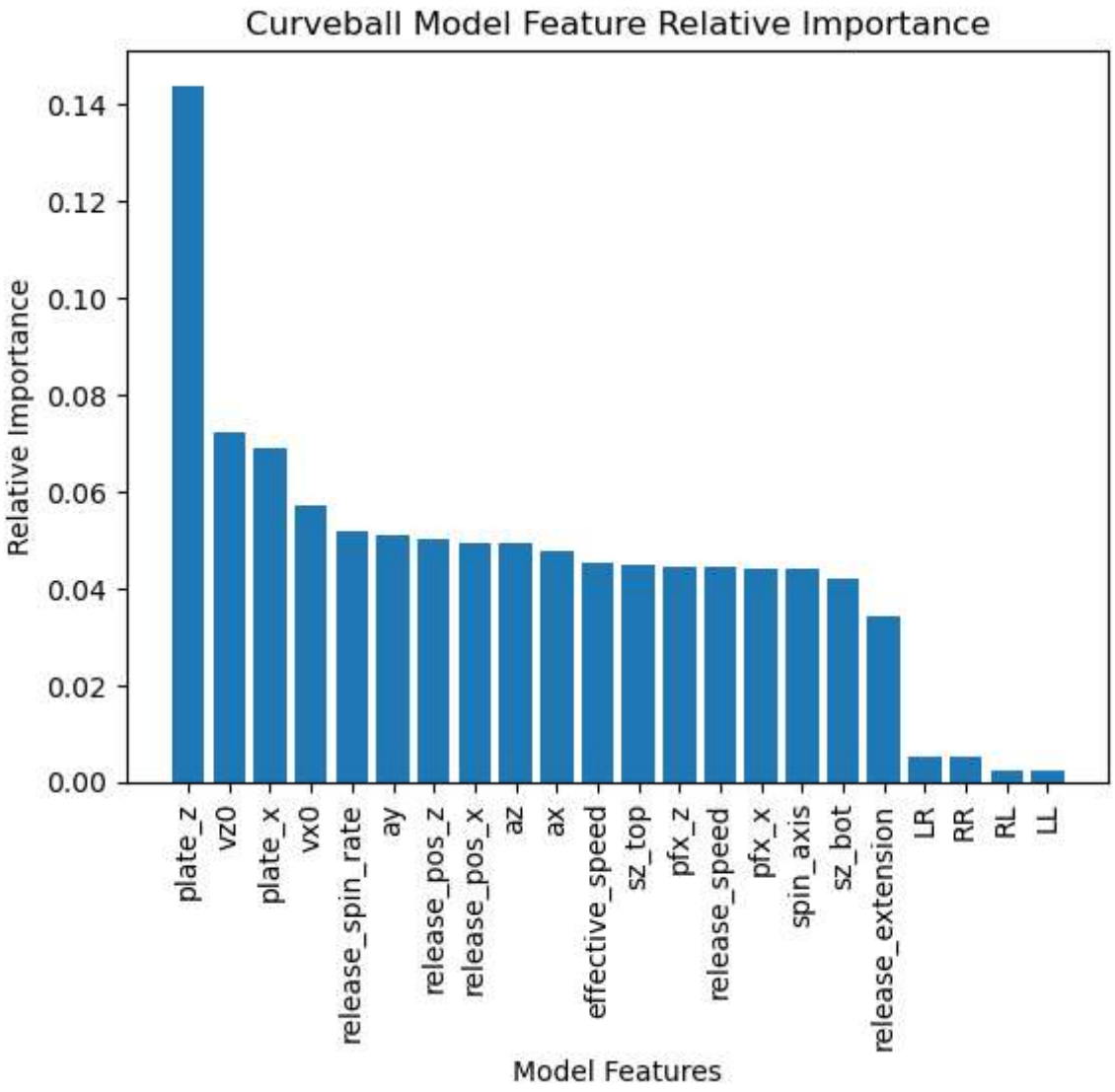


5.2.3. Curveball Model

Like the offspeed model, the curveball model has the same top 3 most important features. What is interesting is the fourth most important feature, "vx0", is the horizontal velocity of the ball. This feature is essentially movement of the ball side-to-side. With this feature being more important compared to the other features, it shows that the developed model values sweeping action on curveballs to generate swings and misses. This encourages pitchers to throw the "sweeping curve" over the traditional "12-6" curve in order to gennerate more misses.

In [13]:
```
curve_mod_imp = curve_tree.feature_importances_

# Curveball Model Feature Importance Plot
sort = np.argsort(curve_mod_imp)[::-1]
curve_s_imp = curve_mod_imp[sort]
curve_s_columns = [curve_data.columns[i] for i in sort]

## Plot Code
plt.bar(range(len(curve_s_imp)), curve_s_imp)
```

```
plt.xlabel('Model Features')
plt.ylabel('Relative Importance')
plt.title('Curveball Model Feature Relative Importance')
plt.xticks(range(len(curve_s_columns)), curve_s_columns, rotation = "vertical")
plt.show()
```

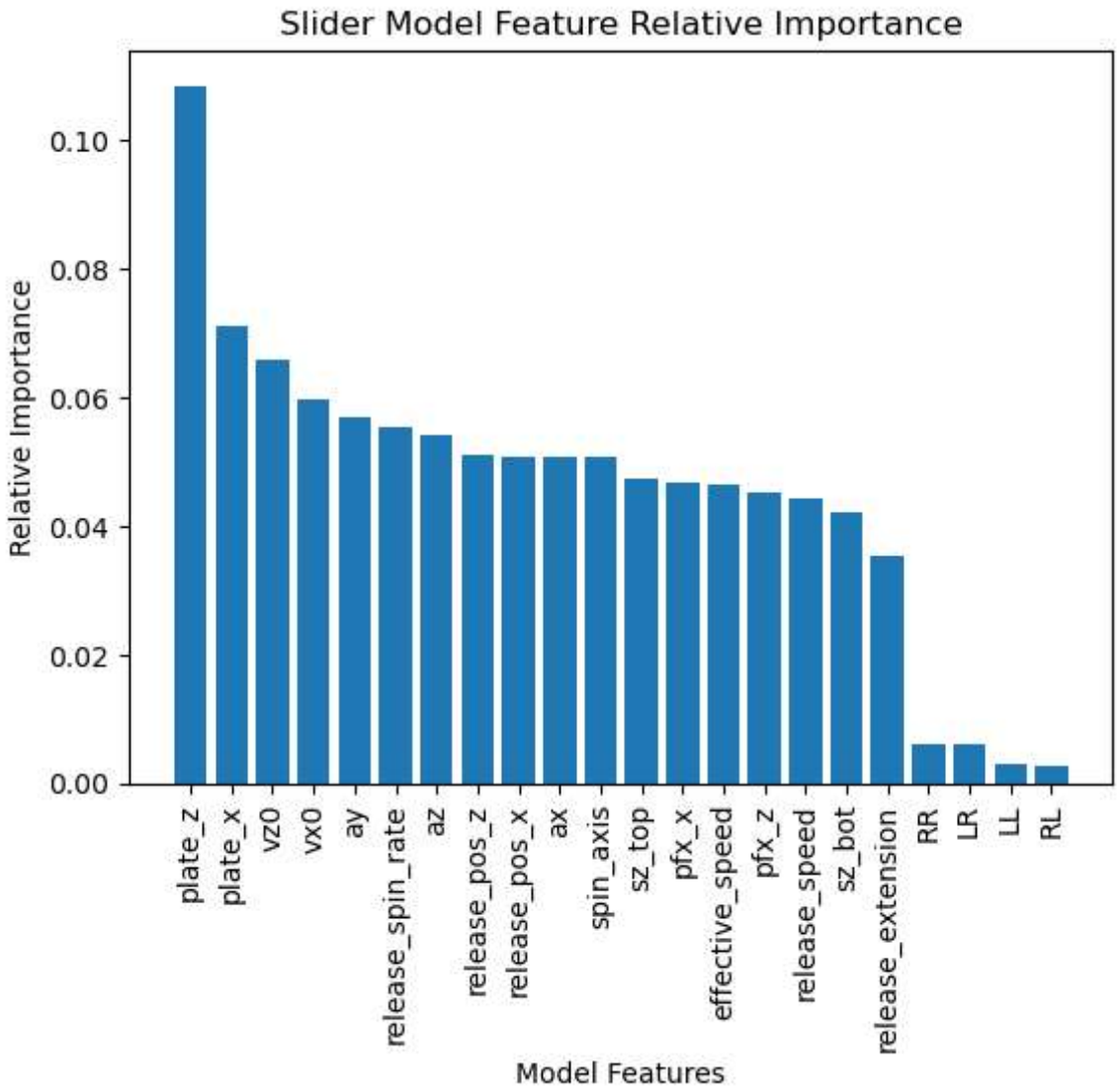


5.2.4. Slider Model

As with the previous three models, the vertical location has the most importance while the horizontal location is second like the fastball model. The vertical and horizontal velocities are third and fourth most important signifying that the movement of the slider is very important in generating swings and misses. This is as expected as sliders generally are effective when they have a lot of movement as well as some other factors such as tunneling which isn't represented in the features explicitly. An interesting note about this model and all the other previous models is the lack of importance the type of matchup has. It was expected for the same hand matchups to hold some sort of relevance in generating a swing miss however, it seems that all the models deemed it was rather insignificant.

In [14]:
```
slid_mod_imp = slid_tree.feature_importances_
```

```
# Slider Model Feature Importance Plot
sort = np.argsort(slid_mod_imp)[::-1]
slid_s_imp = slid_mod_imp[sort]
slid_s_columns = [slid_data.columns[i] for i in sort]

## Plot Code
plt.bar(range(len(fast_s_imp)), fast_s_imp)
plt.xlabel('Model Features')
plt.ylabel('Relative Importance')
plt.title('Slider Model Feature Relative Importance')
plt.xticks(range(len(slid_s_columns)), slid_s_columns, rotation = "vertical")
plt.show()
```



5.3. Predicting Swings and Misses With 2023 Data

Now that the models have been developed, they must now be tested. Each model will be recieving input data from the 2023 season up to the date of this research (6/5/2023). Before finding a player's expected vs actual swing and miss performance, the model predictions must be tested to make sure it is appropriate to apply to expected swing and misses. With these predictions, confusion matrices will be generated for each pitch type model and can be seen below. Using the accuracies for each model, it can be seen that they are all significantly effective

at predicting swings and misses. This is especially evident with the confusion matrices as the models are predicting each category and not just a single category as was the issue without the weights. One thing to note is that the contact prediction accuracies are higher in all models than the miss prediction accuracies. This does not seem to be a problem though as the true swing predictions are still higher across the board than the false miss predictions meaning the models are all still effective at predicting both outcomes. With these results, it is confirmed that all four pitch models are effective at classifying the outcome of a swing and can be moved forward with to rank hitters and pitchers based off their expected as well as actual outcomes.

In [6]:
```
# Importing Metric Modules
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# Preparing Testing Data For Predictions
test_pitches = pd.read_csv('C:/Users/tscot/Downloads/PersonalProjects/swingmissproj/pi

## Creating New Columns For Handedniss
RR = []
LR = []
LL  = []
RL = []
for i in range(test_pitches.shape[0]):
    if test_pitches['stand'][i] == "R" and test_pitches['p_throws'][i] == "R":
        RR.append(1)
        LR.append(0)
        LL.append(0)
        RL.append(0)
    elif test_pitches['stand'][i] == "R" and test_pitches['p_throws'][i] == "L":
        RR.append(0)
        LR.append(0)
        LL.append(0)
        RL.append(1)
    elif test_pitches['stand'][i] == "L" and test_pitches['p_throws'][i] == "R":
        RR.append(0)
        LR.append(1)
        LL.append(0)
        RL.append(0)
    else:
        RR.append(0)
        LR.append(0)
        LL.append(1)
        RL.append(0)

## Removing uneeded columns and adding new columns
test_pitches = test_pitches.iloc[:,1:]
test_pitches = test_pitches.drop(['description', 'des', 'game_pk', 'stand', 'p_throws'
added = {"RR":RR, "LR":LR, "LL":LL, "RL":RL}
test_pitches = pd.concat([test_pitches, pd.DataFrame(added)], axis = 1)

## Seperating The 4 Different Data Sets
### Fastball Test Data
fast_data_test = test_pitches[test_pitches['pitch_type'].isin(['FF', 'FC', 'SI'])]
fast_miss_test = fast_data_test['miss']
fast_data_test = fast_data_test.drop(['miss', 'pitch_type'], axis = 1)
fast_exp = fast_data_test
fast_data_test = fast_data_test.drop(['pitcher_name', 'batter_name'], axis = 1)
```

```
### Offspeed Test Data
off_data_test = test_pitches[test_pitches['pitch_type'].isin(['CH', 'FS'])]
off_miss_test = off_data_test['miss']
off_data_test = off_data_test.drop(['miss', 'pitch_type'], axis = 1)
off_exp = off_data_test
off_data_test = off_data_test.drop(['pitcher_name', 'batter_name'], axis = 1)

### Curveball Test Data
curve_data_test = test_pitches[test_pitches['pitch_type'].isin(['KC', 'CU', 'CS'])]
curve_miss_test = curve_data_test['miss']
curve_data_test = curve_data_test.drop(['miss', 'pitch_type'], axis = 1)
curve_exp = curve_data_test
curve_data_test = curve_data_test.drop(['pitcher_name', 'batter_name'], axis = 1)

### Slider Test Data
slid_data_test = test_pitches[test_pitches['pitch_type'].isin(['SL', 'ST', 'SV'])]
slid_miss_test = slid_data_test['miss']
slid_data_test = slid_data_test.drop(['miss', 'pitch_type'], axis = 1)
slid_exp = slid_data_test
slid_data_test = slid_data_test.drop(['pitcher_name', 'batter_name'], axis = 1)

# Predicting And Testing Accuracy For Each Pitch
## Prediction Formulation
fast_pred = fast_tree.predict(fast_data_test)
off_pred = off_tree.predict(off_data_test)
curve_pred = curve_tree.predict(curve_data_test)
slid_pred = slid_tree.predict(slid_data_test)

## Accuracy Output
print(" Fastball Model: \n \n Confusion Matrix: \n", confusion_matrix(fast_miss_test,
      "\n Accuracy: ", round(accuracy_score(fast_miss_test, fast_pred), 3), "\n \n---
print("Offspeed Model: \n \n Confusion Matrix: \n", confusion_matrix(off_miss_test, of
      "\n Accuracy: ", round(accuracy_score(off_miss_test, off_pred), 3), "\n \n--- \n
print("Curveball Model: \n \n Confusion Matrix: \n", confusion_matrix(curve_miss_test,
      "\n Accuracy: ", round(accuracy_score(curve_miss_test, curve_pred), 3), "\n \n--
print("Slider Model: \n \n Confusion Matrix: \n", confusion_matrix(slid_miss_test, sli
      "\n Accuracy", round(accuracy_score(slid_miss_test, slid_pred), 3))
```

```
 Fastball Model:

 Confusion Matrix:
 [[55366   432]
 [11605   585]]
 Accuracy:  0.823

---

Offspeed Model:

 Confusion Matrix:
 [[11697   486]
 [ 4174   653]]
 Accuracy:  0.726

---

Curveball Model:

 Confusion Matrix:
 [[7444  206]
 [2167  320]]
 Accuracy:  0.766

---

Slider Model:

 Confusion Matrix:
 [[18593  1159]
 [ 5579  2654]]
 Accuracy 0.759
```

6. Ranking Players' 2023 Swing and Miss Performance

This section is broken up into three subsections each grading players differently. The first subsection is ranking hitters on the difference between their expected and actual whiff rate. The second subsection ranks pitchers on their expected whiff rate while the third subsection ranks pitchers on the difference between their expected and actual whiff rate.

6.1. Offensive Expected vs. Actual Performance

This subsection ranks a hitter based off the difference between their expected and actual whiff rates. If a hitter whiffs less than expected then they will be ranked higher while vice versa results in a lower ranking. The top 10 hitters and their performance against each pitch will be shown as well as the worst 10 hitters. With these rankings, a visualization of the differences will be shown with some of the most noteable hitters labeled for compare and contrast. The intersting thing about these visualizations is that some of the most prominent hitters don't necessarily live up to the hype and begs the question whether the public eye overvalues their abilities. In the discussion portions of this subsection the difference between their expected and actual whiff rates will also be referenced as "difference score".

6.1.1. Fastball Model Rankings

Below are the aforementioned top 10 and worst 10 hitters in terms of expected vs. actual whiff rates on fastballs so far in the 2023 season. To be eligible, a batter has to have seen over 100 fastballs so far this season. The rankings on the left are the best contact hitters according to the model while the right are the worst. Essentially, the higher the difference, the worse they performed. Some of the most notorious contact hitters crack the top of the list while some of the most notorious non-contact hitters appear at the bottom. Especially on the top side of the rankings, there isn't really a suprise performer. Looking at the worst hitters, there are some relatively unkown names however this is to be expected as these are the "worst" hitters. They aren't expected to have notoriety around their name.

In [7]:
```
# Loading Needed Modules
from IPython.display import display

# Fastball Calculations
fast_exp['expected_difference_score'] = (fast_miss_test - fast_pred)
fast_batter_names = fast_exp['batter_name'].unique()
fast_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(fast_batter_names)):
    batter = fast_exp[fast_exp['batter_name'] == fast_batter_names[i]]
    if batter.shape[0] > 100:
        values = {'player_name': fast_batter_names[i], 'score': batter['expected_diffe
                                'pitches_count': batter.shape[0]}
        fast_score_frame = pd.concat([fast_score_frame, pd.DataFrame(values, index = |
                                      ignore_index = True)
fast_score_frame = fast_score_frame.reset_index(drop = True)

# Rankings Output
pd.concat([fast_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].re
           fast_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].r
```

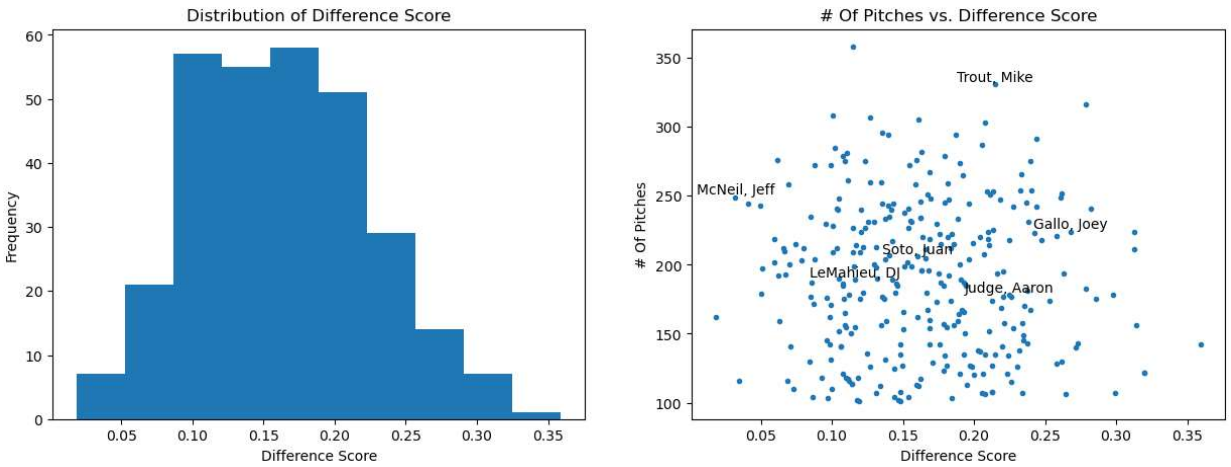
Out[7]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Ruiz, Keibert	0.018519	162	Zunino, Mike	0.359155	142
1	McNeil, Jeff	0.032129	249	Siri, Jose	0.319672	122
2	Sheets, Gavin	0.034483	116	Thaiss, Matt	0.319672	122
3	Arraez, Luis	0.040984	244	Maldonado, Martín	0.314103	156
4	Hoerner, Nico	0.049383	243	Outman, James	0.312796	211
5	Vierling, Matt	0.050279	179	Noda, Ryan	0.312500	224
6	Verdugo, Alex	0.050761	197	Doyle, Brenton	0.299065	107
7	Rutschman, Adley	0.059361	219	Henderson, Gunnar	0.297753	178
8	Estrada, Thairo	0.059406	202	Rosario, Eddie	0.285714	175
9	Kwan, Steven	0.061594	276	Rooker, Brent	0.282158	241

The below two plots show the distribution of the difference scores as well as a scatter plot of the difference scores of eligible hitters. There are some of the most reputable hitters charted in

order to see where they fall amongst the league. As seen below, Juan Soto despite being recognized as one of the best pure hitters in the league, is relatively average in terms of his performance. He did have a down year last year and hasn't quite returned to form in 2023 but, it is intriguing to seem in the middle of the pack. Another noteable player is Mike Trout, one of the greatest players of all time if not the greatest, is ranked below average in comparison to his peers. It should be noted that he has seen the second most fastballs in the league this year and could potentially signify a relative weakness against the pitch and why he has seen so many.

In [8]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(fast_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(fast_score_frame['score'], fast_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Soto, Juan', 'Judge, Aaron', 'Trout, Mike', 'McNeil, Jeff', 'LeMahieu, DJ'
location = np.where(np.isin(fast_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(fast_score_frame['score'].iloc[loc], fast_score_frame['pitches_count'].il

# Output Call
plt.show()
```



6.1.2. Offspeed Model Rankings

Next we look at the top 10 and worst 10 hitters in terms of the difference between expected and actual whiff rate on offspeed pitches so far in the 2023 season. The output of these rankings are the same style as the fastball section above and will follow the same format for the remaining rankings in this paper. To be eligible, a batter must have faced more than 25 offspeed

pitches so far this season. These rankings are much more intersting and unexpected, especially the top performing hitters. Eric Hosmer is widely regarded as past his prime however, is the best performing contact hitter against off speed pitches. As for the rest of the top ranked guys, Bo Bichette and Alec Bohm are the only names with some degree of notoriety while the other guys are relatively unkown. Due to the sample size being lower than the other pitches, these results should be taken with a grain of salt however, it is very intersting to see the top guys. What seems to validate these rankings are the bottom guys as many of them are notorious for being all or nothing hitters such as Joey Gallo and Luke Voit. With these guys being properly ranked, it supports the effeciveness of the offspeed model and supports the top guys for being where they are at, potentially suggesting they are undervalued amongst the baseball community.

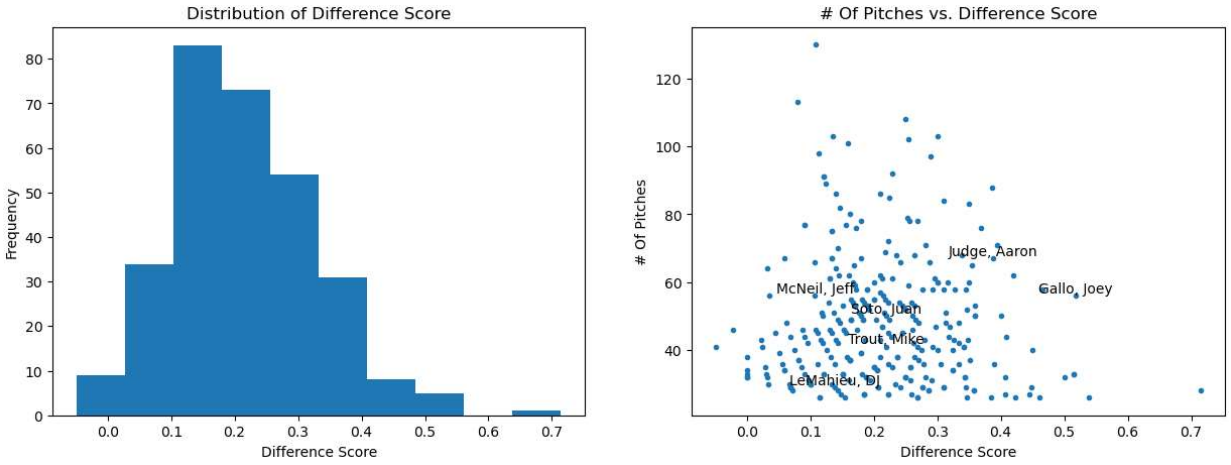
In [9]:
```
# Offspeed Calculations
off_exp['expected_difference_score'] = (off_miss_test - off_pred)
off_batter_names = off_exp['batter_name'].unique()
off_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(off_batter_names)):
    batter = off_exp[off_exp['batter_name'] == off_batter_names[i]]
    if batter.shape[0] > 25:
        values = {'player_name': off_batter_names[i], 'score': batter['expected_differ
                            'pitches_count': batter.shape[0]}
        off_score_frame = pd.concat([off_score_frame, pd.DataFrame(values, index = [0
                                    ignore_index = True)

# Rankings Output
pd.concat([off_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].res
           off_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].re
```

Out[9]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Hosmer, Eric	-0.048780	41	Voit, Luke	0.714286	28
1	Vargas, Miguel	-0.021739	46	Garrett, Stone	0.538462	26
2	Escobar, Eduardo	0.000000	33	Gallo, Joey	0.517857	56
3	Vierling, Matt	0.000000	34	Brown, Seth	0.515152	33
4	Bichette, Bo	0.000000	32	Diaz, Jordan	0.515152	33
5	Dubón, Mauricio	0.000000	38	Julien, Edouard	0.500000	32
6	Refsnyder, Rob	0.000000	32	Maton, Nick	0.465517	58
7	Burleson, Alec	0.023256	43	Montero, Elehuris	0.461538	26
8	Bohm, Alec	0.024390	41	Taylor, Chris	0.450000	40
9	Brennan, Will	0.028571	35	McKenna, Ryan	0.448276	29

Looking at the visuals, we can see that the distribution of differences is roughly normal while our stars we mentioned previously performed better compared to the rest of the league. The notorious swing and miss guys like Judge and Gallo are towards the bottom while Soto and Trout climbed in the comparisons. This suggests that Soto and Trout are better against these offspeed pitches in comparison to seeing fastballs.

In [10]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(off_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(off_score_frame['score'], off_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Soto, Juan', 'Judge, Aaron', 'Trout, Mike', 'McNeil, Jeff', 'LeMahieu, DJ'
location = np.where(np.isin(off_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(off_score_frame['score'].iloc[loc], off_score_frame['pitches_count'].iloc

# Output Call
plt.show()
```



6.1.3. Curveball Model Rankings

In the below section, the top 10 best and worst hitters against curveballs so far for the 2023 season are shown. To be eligble for the presented rankings, a hitter must have seen 15 curveballs at least this season. The results are very interesting as there are some unexpected as well as expected names in both the best and worst columns. For example, Mike Trout is ranked as one of the top 10 worst hitters in baseball so far in the 2023 season against curveballs due to his lack of performance compared to his expected performance. While his presence is suprising, other names in the bottom of the list don't suprise as much such as Mike Zunino, Jose Siri, and Ryan Noda who have all been present in the prior rankings listed. Looking at the top hitters against curveballs this year, there aren't any superstars present however known contact hitters are present so there aren't any incredible suprises. An intersting name present on the list is Martin Maldonado who was present in the fastball rankings however, on the worst 10 side. It is

intriguing to see a player be polar opposites in performance based off of which pitch they are seeing.

In [11]:
```
# Curveall Calculations
curve_exp['expected_difference_score'] = (curve_miss_test - curve_pred)
curve_batter_names = curve_exp['batter_name'].unique()
curve_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(curve_batter_names)):
    batter = curve_exp[curve_exp['batter_name'] == curve_batter_names[i]]
    if batter.shape[0] > 15:
        values = {'player_name': curve_batter_names[i], 'score': batter['expected_diff
                                'pitches_count': batter.shape[0]}
        curve_score_frame = pd.concat([curve_score_frame, pd.DataFrame(values, index =
                                       ignore_index = True)

# Rankings Output
pd.concat([curve_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].r
           curve_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,]
```

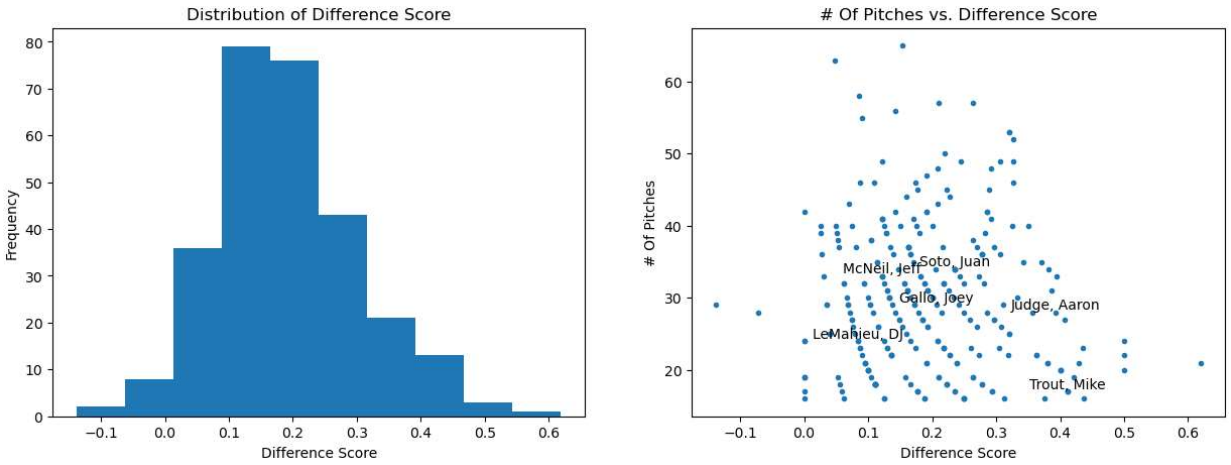
Out[11]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Kim, Ha-Seong	-0.137931	29	Villar, David	0.619048	21
1	Burleson, Alec	-0.071429	28	Siri, Jose	0.500000	20
2	Turner, Justin	0.000000	19	Zunino, Mike	0.500000	24
3	Smith, Will	0.000000	24	Noda, Ryan	0.500000	22
4	Díaz, Yandy	0.000000	24	Vientos, Mark	0.437500	16
5	Kemp, Tony	0.000000	19	Bradley Jr., Jackie	0.434783	23
6	Maldonado, Martín	0.000000	17	Myers, Wil	0.428571	21
7	Escobar, Eduardo	0.000000	16	Hilliard, Sam	0.421053	19
8	Tapia, Raimel	0.000000	19	Vosler, Jason	0.411765	17
9	Benintendi, Andrew	0.000000	42	Langeliers, Shea	0.411765	17

The curveball difference score distribution follows a roughly normal shape. Looking at the well know players in the league, there aren't any suprises besides the previously mentioned Mike Trout difference score. While Judge is on the lesser side and seems to have been barely left off the worst 10 hitters list, he isn't a pure contact hitter and while hits for power at a high level, does have his fair share of swings and misses. The reason why it is suprising to see Trout as one of the worst hitters against curveballs is that he is regarded as an all around great hitter.

In [12]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(curve_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")
```

```
# Plot 2
sub[1].plot(curve_score_frame['score'], curve_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Soto, Juan', 'Judge, Aaron', 'Trout, Mike', 'McNeil, Jeff', 'LeMahieu, DJ'
location = np.where(np.isin(curve_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(curve_score_frame['score'].iloc[loc], curve_score_frame['pitches_count'],

# Output Call
plt.show()
```



6.1.4. Slider Model Rankings

Of the four pitch models, the slider model rankings is last to be shown which can be seen in below figure. This shows the player rankings of the difference between the expected model whiff rate and actual whiff rate for sliders in the 2023 season. To be eligible, players must have seen 40 sliders up to the point in the season when the testing data was gathered. The slider model rankings show interesting results with Aaron Judge being the second worst player against sliders in 2023. Despite his MVP status, he whiffs at an extremely high rate in comparison to what is expected of him based off the type of sliders he is seeing. As for the top ranked players in terms of contact against sliders in 2023, there are some reoccurring names such as Keibert Ruiz or Bo Bichette. This type of frequent presence across the different pitch models was expected at the beginning of this research as it was expected for good contact hitters to do well across all pitches. This idea will be discussed more in the next subsection comparing players against all pitches however, was to be noted here to notice the occassional repeated names.

In [13]:
```
# Slider Calculations
slid_exp['expected_difference_score'] = (slid_miss_test - slid_pred)
slid_batter_names = slid_exp['batter_name'].unique()
slid_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(slid_batter_names)):
    batter = slid_exp[slid_exp['batter_name'] == slid_batter_names[i]]
    if batter.shape[0] > 40:
```

```
        values = {'player_name': slid_batter_names[i], 'score': batter['expected_diffe
                                 'pitches_count': batter.shape[0]}
        slid_score_frame = pd.concat([slid_score_frame, pd.DataFrame(values, index = |
                                      ignore_index = True)

# Rankings Output
pd.concat([slid_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].re
           slid_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].r
```

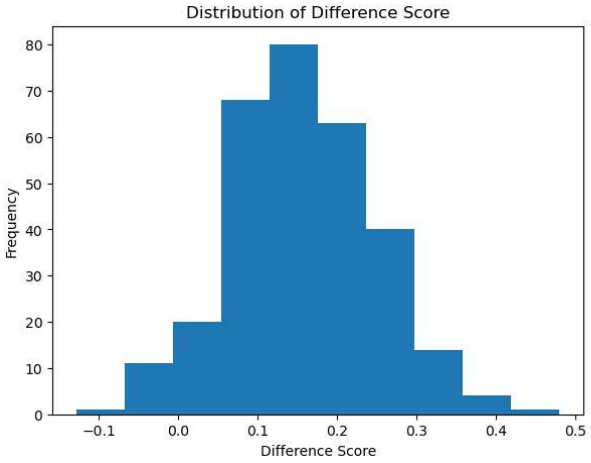
Out[13]:

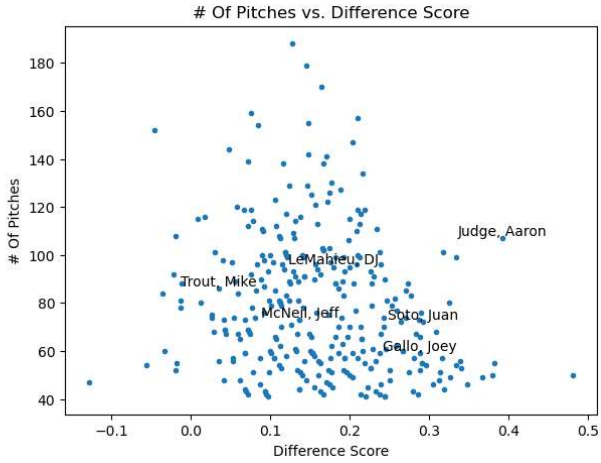
	player_name	score	pitches_count	player_name	score	pitches_count
0	Gurriel, Yuli	-0.127660	47	Larnach, Trevor	0.480000	50
1	Straw, Myles	-0.055556	54	Judge, Aaron	0.392523	107
2	Bichette, Bo	-0.046053	152	Grossman, Robbie	0.381818	55
3	Hoerner, Nico	-0.035714	84	Castro, Rodolfo	0.380000	50
4	Ruiz, Keibert	-0.033333	60	Fairchild, Stuart	0.367347	49
5	Díaz, Aledmys	-0.021739	92	Cabrera, Miguel	0.347826	46
6	Gonzalez, Oscar	-0.019231	52	Bae, Ji Hwan	0.339623	53
7	Dubón, Mauricio	-0.018519	108	Crawford, Brandon	0.339286	56
8	Kiner-Falefa, Isiah	-0.018182	55	Sabol, Blake	0.333333	54
9	Bohm, Alec	-0.012821	78	Martinez, J.D.	0.333333	99

Once again, the distribution of the difference scores is roughly normal with no distinct skew. Looking at the plot of difference score and amount of pitches, our noteable stars have shifted around in comparison to the rest of the league. Mike Trout against sliders is one of the top performers against sliders so far this year even though he was the second worst against curveballs. Juan Soto despite being well regarded as an astronomically better "pure hitter" than Joey Gallo has essentially the same difference score against sliders. Jeff McNeil and D.J. LeMahuieu are also both intersting cases as they are both pedestrian in comparison to the rest of the league.

In [14]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(slid_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(slid_score_frame['score'], slid_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Soto, Juan', 'Judge, Aaron', 'Trout, Mike', 'McNeil, Jeff', 'LeMahieu, DJ'
```

```
location = np.where(np.isin(slid_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(slid_score_frame['score'].iloc[loc], slid_score_frame['pitches_count'].i

# Output Call
plt.show()
```





6.1.5. Combined Models Rankings

In order to rank overall contact performance of hitters for the 2023 season, all the four generated pitch models will be combined. Essentially the difference score for a hitter against each pitch is put together and waited based off how many times they have seen that pitch. The combined rankings heavily resemble the fastball model rankings which is due to the reason that the majority of pitches that hitters see are fastballs. With that being said, they don't exactly reflect the fastball model rankings for the 2023 season. Looking at the actual rankings themselves, the top ranked hitters are filled more so than other individual models with well known names. There are no suprises at top of the rankings. Guys like Keibert Ruiz and Luis Arraez aren't household names but are well regarded as very potent contact hitters who don't swing and miss a lot. Luis Arraez in particular is hitting around .400 this year and surely is not a suprise to see that he is making contact at a much higher rate than expected. As for the worst contact hitters according to all four models, most names have appeared as the worst hitters for other pitches as well. With that being said, Bryce Harper being ranked as one of the worst hitters is a suprise because he isn't present in the worst 10 rankings of any individual models. Harper isn't known as a contact hitter but, isn't known as being a swing and miss guy which makes this ranking even more suprising. This shows that he isn't particularly great contact wise against any of the pitches and so when they were combined he performed worse overall in comparison to other players who struggled against one particular pitch.

In [15]:
```
# Total
total_frame = pd.DataFrame(columns = ['player_name', 'score', 'total_pitches'])
batter_names = fast_exp['batter_name'].unique()
for i in range(len(batter_names)):
    fast = fast_exp[fast_exp['batter_name'] == batter_names[i]]
    off = off_exp[off_exp['batter_name'] == batter_names[i]]
    curve = curve_exp[curve_exp['batter_name'] == batter_names[i]]
    slid = slid_exp[slid_exp['batter_name'] == batter_names[i]]
```

```
    if fast.shape[0] > 100 and off.shape[0] > 25:
        if curve.shape[0] > 15 and slid.shape[0] > 40:
            total_count = fast.shape[0] + off.shape[0] + curve.shape[0] + slid.shape[0
            score_calc = (fast.shape[0]/total_count) * (fast['expected_difference_scor

            values = {'player_name': batter_names[i], 'score': score_calc, 'total_pitc
            total_frame = pd.concat([total_frame, pd.DataFrame(values, index = [0])],

# Rankings Output
pd.concat([total_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].reset_i
           total_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].reset_
```

Out[15]:

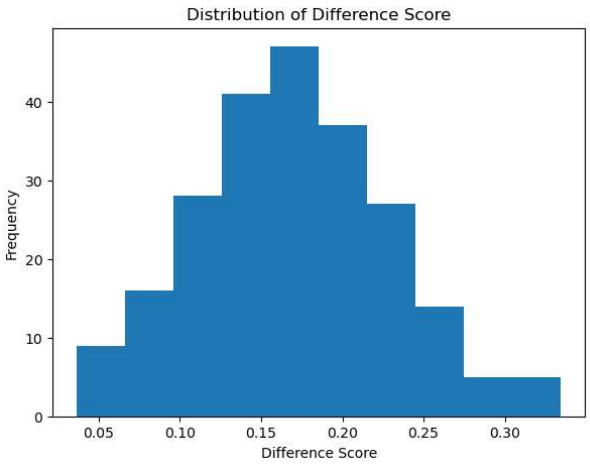
	player_name	score	total_pitches	player_name	score	total_pitches
0	Hoerner, Nico	0.036408	412	Zunino, Mike	0.334646	254
1	Arraez, Luis	0.042289	402	Larnach, Trevor	0.325843	267
2	Ruiz, Keibert	0.046440	323	Noda, Ryan	0.322857	350
3	Burleson, Alec	0.047059	255	Sabol, Blake	0.319298	285
4	Vierling, Matt	0.049505	303	Judge, Aaron	0.313158	380
5	Díaz, Aledmys	0.062992	254	Gallo, Joey	0.304348	368
6	Gurriel Jr., Lourdes	0.062992	381	Harper, Bryce	0.295547	247
7	Franco, Wander	0.064748	417	Outman, James	0.289474	380
8	Bichette, Bo	0.065517	580	Taylor, Chris	0.287162	296
9	Dubón, Mauricio	0.066667	375	Rooker, Brent	0.282878	403

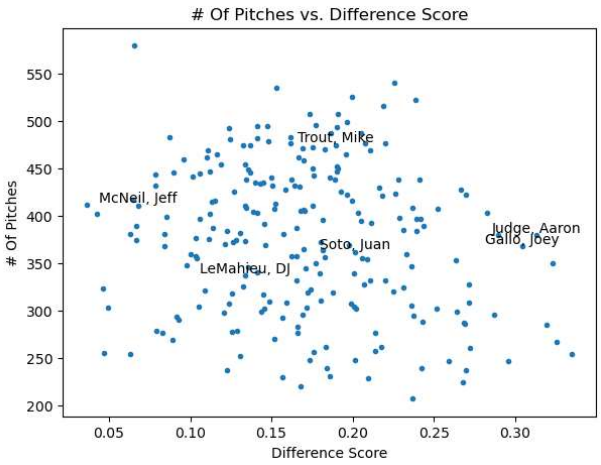
The below distribution of all the difference scores follows a normal distribution tigher than all the previous individual models. As for the scatter plot with notable names, there are no suprises present that were not already expected or noted. Once again, Aaron Judge and Joey Gallo are at the bottom echelon in terms of contact performance judged by the models while Jeff McNeil is noticeably better than league average performance.

In [16]:

```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(total_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(total_frame['score'], total_frame['total_pitches'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Soto, Juan', 'Judge, Aaron', 'Trout, Mike', 'McNeil, Jeff', 'LeMahieu, DJ'
location = np.where(np.isin(total_frame['player_name'], players))
```

```
for loc in location[0]:
    plt.text(total_frame['score'].iloc[loc], total_frame['total_pitches'].iloc[loc], t

# Output Call
plt.show()
```





All in all, these types of rankings are able to remove bias when judging player performance as well as give an opportunity for lesser know players to be recognized for their contact abilities based off of the degree of difficulty of the pitches they are seeing. These types of metrics and rankings are able to capture player capability more effectively than just going off of a blanket whiff rate as it gives an opportunity to quantify the difficulty of the actual pitches a player is seeing.

6.2. Ranking Pitchers Off Expected Performance

This subsection ranks pitchers based off their expected whiff rate. In comparison to the section before, pitchers will get ranked off the expected whiff rate of that particular pitch. These pitchers are essentially being ranked off whether the pitch metrics of the pitches they threw were expected to generate a miss when swung at. Each pitch is thrown differently and has different attributes amongst pitchers. By ranking players by their expected whiff rate, we can not only remove the luck more effectively when quantifying player ability but, learn key pitch design insights from which qualities of the certain pitch generate misses more frequently.

6.2.1. Fastball Model Rankings

Below are the top 10 pitchers and top 10 worst pitchers in expected whiff rate on fastballs for the 2023 season. Each pitcher must have thrown 90 fastballs in order to qualify for these rankings. These expected values are generated from the pitches thrown but, do not represent any of the actual outcomes only what is expected by the model. Looking at the expected whiff rates and the players who hold them, it is clear that for fastballs the model values movement over velocity. Of the guys who are in the top of the expected whiffs, none are exceptional flame throwers and most are the opposite and generate misses through movement. Luis Garcia for example, utilizes a cutter more predominantly than his 4-seam fastball. This is rather unexpected as I would have assumed before the model fitting process that velocity would be

more valueable at generating a swing and a miss than movement. This isn't a huge suprise as the use of pitches like a sinker and cutter have become more popular over the years. As for the lowest expected whiff rates, there is one suprise name, Joe Musgrove. Musgrove is one of the better pitchers in the league and throws both a cutter and sinker which are known to be effective. It is interesting that the fastball model doesn't consider his pitches efficient in creating swings and misses.

In [17]:
```
# Pitcher Rankings
fast_exp['expected_score'] = fast_pred
fast_pitcher_names = fast_exp['pitcher_name'].unique()
fast_score_frame = pd.DataFrame(columns = ['player_name', 'expected_score', 'pitches_c
for i in range(len(fast_pitcher_names)):
    pitcher = fast_exp[fast_exp['pitcher_name'] == fast_pitcher_names[i]]
    if pitcher.shape[0] > 90:
        values = {'player_name': fast_pitcher_names[i], 'expected_score': pitcher['exp
                                'pitches_count': pitcher.shape[0]}
        fast_score_frame = pd.concat([fast_score_frame, pd.DataFrame(values, index = |
                                      ignore_index = True)
fast_score_frame = fast_score_frame.reset_index(drop = True)

# Rankings Output
pd.concat([fast_score_frame.sort_values(by = 'expected_score', ascending = False).iloc
           fast_score_frame.sort_values(by = 'expected_score', ascending = True).iloc|
```

Out[17]:

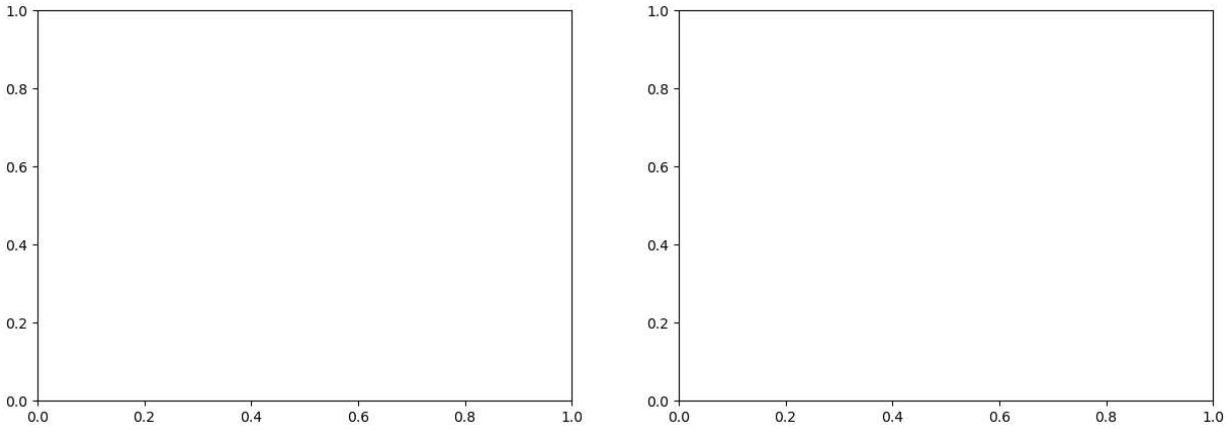
	player_name	expected_score	pitches_count	player_name	expected_score	pitches_count
0	Garcia, Luis	0.153005	183	Germán, Domingo	0.0	150
1	Velasquez, Vince	0.064516	124	Yarbrough, Ryan	0.0	114
2	Wantz, Andrew	0.057143	105	Ragans, Cole	0.0	108
3	Bednar, David	0.056911	123	Ginkel, Kevin	0.0	105
4	Bieber, Shane	0.056537	283	Staumont, Josh	0.0	99
5	Crawford, Kutter	0.054726	201	Jiménez, Joe	0.0	100
6	Rucker, Michael	0.054422	147	Harvey, Hunter	0.0	131
7	Gallen, Zac	0.053097	339	Musgrove, Joe	0.0	164
8	Minter, A.J.	0.050000	200	Vest, Will	0.0	92
9	Javier, Cristian	0.049451	364	Cimber, Adam	0.0	102

The distribution of expected whiff rates for the fastballs thrown in 2023 is not normal and follows more of a Log-normal distribution. This becomes more evident when looking at the scatter plot of expected whiff rates of all qualifying pitchers. Some of the top fastball pitchers are plotted by name to give reference and to see how they compare to the rest of the league. Looking at the plot, many of the qualified fastball pitchers have a very low expected whiff rate on their fastballs. This is again related to the idea that the model values movement over velocity while many of these type guys rely on their velocity to get swings and misses. This could

potentially be a limitation and will be discussed further in depth when comparing the actual vs expected whiff rates of pitchers and their pitches.

In [33]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(fast_score_frame['expected_score'])
sub[0].set_title("Distribution of Exptected Whiff Rate")
sub[0].set_xlabel("Expected Whiff Rate")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(fast_score_frame['expected_score'], fast_score_frame['pitches_count'], '.'
sub[1].set_title("# Of Pitches vs. Expected Whiff Rate")
sub[1].set_xlabel("Expected Whiff Rate")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Cole, Gerrit', 'Bieber, Shane', 'Alcantara, Sandy', 'Verlander, Justin', '
location = np.where(np.isin(fast_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(fast_score_frame['expected_score'].iloc[loc], fast_score_frame['pitches_c

# Output Call
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3802, in Index.get_loc
(self, key, method, tolerance)
   3801 try:
-> 3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:

File ~\anaconda3\lib\site-packages\pandas\_libs\index.pyx:138, in pandas._libs.index.
IndexEngine.get_loc()

File ~\anaconda3\lib\site-packages\pandas\_libs\index.pyx:165, in pandas._libs.index.
IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObject
HashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObject
HashTable.get_item()

KeyError: 'expected_score'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[33], line 6
      3 figure, sub = plt.subplots(1,2, figsize = (15,5))
      5 ## Plot 1
----> 6 sub[0].hist(fast_score_frame['expected_score'])
      7 sub[0].set_title("Distribution of Exptected Whiff Rate")
      8 sub[0].set_xlabel("Expected Whiff Rate")

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:3807, in DataFrame.__getitem_
_(self, key)
   3805 if self.columns.nlevels > 1:
   3806     return self._getitem_multilevel(key)
-> 3807 indexer = self.columns.get_loc(key)
   3808 if is_integer(indexer):
   3809     indexer = [indexer]

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3804, in Index.get_loc
(self, key, method, tolerance)
   3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:
-> 3804     raise KeyError(key) from err
   3805 except TypeError:
   3806     # If we have a listlike key, _check_indexing_error will raise
   3807     #  InvalidIndexError. Otherwise we fall through and re-raise
   3808     #  the TypeError.
   3809     self._check_indexing_error(key)

KeyError: 'expected_score'
```



6.2.2. Offspeed Model Rankings

The next subsection looks into the 10 top and worst expected whiff rates from pitchers on offspeed pitches in the 2023 season. To be eligible pitchers must have thrown more than 20 offspeed pitches so far this season. The generated expected whiff rate rankings align more with expectations as more of the top pitchers are present at the top of the list. Pitchers like Max Fried and Kevin Gausman are well known as having spectacular changeups and the expected whiff rate rankings reflect this. The pitchers with the worst expected whiff rates also make sense as guys like Michael Kopech are effective due to throwing hard not for their offspeed pitch.

In [19]:
```
# Offspeed Calculations
off_exp['expected_score'] = off_pred
off_pitcher_names = off_exp['pitcher_name'].unique()
off_score_frame = pd.DataFrame(columns = ['player_name', 'expected_score', 'pitches_co
for i in range(len(off_pitcher_names)):
    pitcher = off_exp[off_exp['pitcher_name'] == off_pitcher_names[i]]
    if pitcher.shape[0] > 20:
        values = {'player_name': off_pitcher_names[i], 'expected_score': pitcher['expe
                            'pitches_count': pitcher.shape[0]}
        off_score_frame = pd.concat([off_score_frame, pd.DataFrame(values, index = [0
                                    ignore_index = True)

# Rankings Output
pd.concat([off_score_frame.sort_values(by = 'expected_score', ascending = False).iloc[
           off_score_frame.sort_values(by = 'expected_score', ascending = True).iloc[0
```

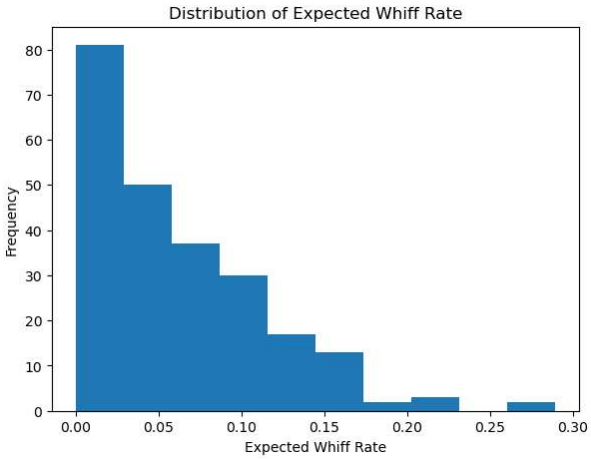
Out[19]:

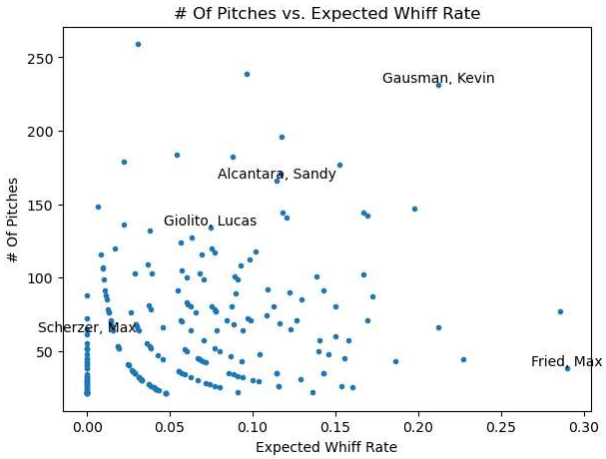
	player_name	expected_score	pitches_count	player_name	expected_score	pitches_count
0	Fried, Max	0.289474	38	Lynch, Daniel	0.0	26
1	Williams, Devin	0.285714	77	Blach, Ty	0.0	29
2	Neris, Hector	0.227273	44	Barria, Jaime	0.0	38
3	Gausman, Kevin	0.212121	231	Milner, Hoby	0.0	26
4	Miley, Wade	0.212121	66	Pagán, Emilio	0.0	22
5	McClanahan, Shane	0.197279	147	Kopech, Michael	0.0	21
6	Corbin, Patrick	0.186047	43	Gibaut, Ian	0.0	22
7	Brazoban, Huascar	0.172414	87	Moreta, Dauri	0.0	32
8	Greinke, Zack	0.169014	71	Kaprielian, James	0.0	34
9	Kelly, Merrill	0.169014	142	Kirby, George	0.0	24

The expected whiff rate distribution is similar to the fastball distribution where it follows more of a Log-normal distribution. There are more expected whiff rates away from 0 than the fastball model however, still not normally distributed. Looking at the more well know pitchers plotted in the scatter plot, many of them have higher expected whiff rates than the rest of the league meaning the metrics and pitch design of their offspeed pitches match their notoriety.

In [20]:

```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(off_score_frame['expected_score'])
sub[0].set_title("Distribution of Expected Whiff Rate")
sub[0].set_xlabel("Expected Whiff Rate")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(off_score_frame['expected_score'], off_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Expected Whiff Rate")
sub[1].set_xlabel("Expected Whiff Rate")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names

players = ['Gausman, Kevin', 'Giolito, Lucas', 'Alcantara, Sandy', 'Fried, Max', 'Sche
location = np.where(np.isin(off_score_frame['player_name'], players))
for loc in location[0]:
    plt.text(off_score_frame['expected_score'].iloc[loc], off_score_frame['pitches_cou

# Output Call
plt.show()
```





6.2.3. Curveball Model Rankings

Below are the top and worst expected whiff rates by a player for curveballs in the 2023 season. To qualify to be on these lists a pitcher must have thrown over 10 curveballs this year. The names on both lists are intersting as the model seems to classify some of the more noteable curveballs in the game such as Charlie Morton and Framber Valdez however, Clayton Kershaw is on the worst expected whiff rates list. Kershaw has one of the most noteable curveballs in the league while the model seems to support the sentiment that it is heavily overvalued by the public. This type of disagreement between model and public sentiment will be intersting to come back to in the next section when comparing the difference between a pitcher's expected and actual whiff rate for these pitches. This could potentially be a limitation to the model as it failed to capture certain effectiveness features of a curveball however, it could also represent a lack of understanding of true ability by the average fan.

In [21]:
```
# Curveall Calculations
curve_exp['expected_score'] = curve_pred
curve_pitcher_names = curve_exp['pitcher_name'].unique()
curve_score_frame = pd.DataFrame(columns = ['player_name', 'expected_score', 'pitches_
for i in range(len(curve_pitcher_names)):
    pitcher = curve_exp[curve_exp['pitcher_name'] == curve_pitcher_names[i]]
    if pitcher.shape[0] > 10:
        values = {'player_name': curve_pitcher_names[i], 'expected_score': pitcher['ex
                                 'pitches_count': pitcher.shape[0]}
        curve_score_frame = pd.concat([curve_score_frame, pd.DataFrame(values, index =
                                       ignore_index = True)

# Rankings Output
pd.concat([curve_score_frame.sort_values(by = 'expected_score', ascending = False).ilo
           curve_score_frame.sort_values(by = 'expected_score', ascending = True).iloc
```

Out[21]:

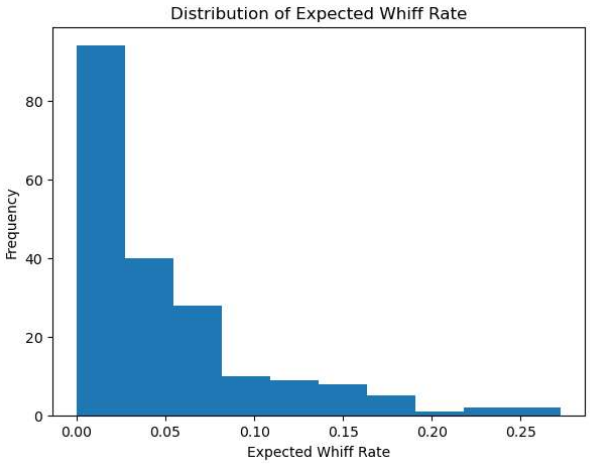
	player_name	expected_score	pitches_count	player_name	expected_score	pitches_count
0	Sands, Cole	0.272727	11	Syndergaard, Noah	0.0	40
1	Mantiply, Joe	0.250000	16	Woodford, Jake	0.0	26
2	Cobb, Alex	0.240000	50	Faucher, Calvin	0.0	15
3	Morton, Charlie	0.222727	220	Kershaw, Clayton	0.0	92
4	Barlow, Scott	0.200000	45	Ragans, Cole	0.0	17
5	Ramírez, Erasmo	0.181818	11	Bumgarner, Madison	0.0	36
6	Valdez, Framber	0.181818	121	Bubic, Kris	0.0	21
7	Lodolo, Nick	0.168067	119	Manning, Matt	0.0	20
8	Zastryzny, Rob	0.166667	30	Pivetta, Nick	0.0	116
9	Hernández, Carlos	0.166667	18	Kuhl, Chad	0.0	32

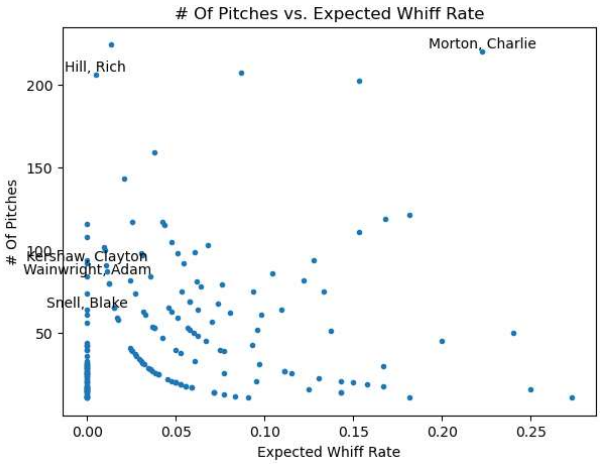
Just like the previous distributions for pitches, the curveball model expected whiff rates follow a Log-normal distribution more than a normal distribution. The scatter plot with the names of players with iconic curveballs shows an intersting message. Many of these well know curveball throwers have expected whiff rates of 0 or a value close to 0. This again represents the idea that there is a lapse either between the public sentiment of the effectiveness of these players' curveballs or the model failed to capture the effective metrics of curveballs. With this being said, Charlie Morton is highly valued by the curveball model and seems to support the notion that the other players may be overvalued. This idea will be investigated more in the next section.

In [22]:

```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(curve_score_frame['expected_score'])
sub[0].set_title("Distribution of Expected Whiff Rate")
sub[0].set_xlabel("Expected Whiff Rate")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(curve_score_frame['expected_score'], curve_score_frame['pitches_count'],
sub[1].set_title("# Of Pitches vs. Expected Whiff Rate")
sub[1].set_xlabel("Expected Whiff Rate")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Kershaw, Clayton', 'Wainwright, Adam', 'Morton, Charlie', 'Hill, Rich', 'S
location = np.where(np.isin(curve_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(curve_score_frame['expected_score'].iloc[loc], curve_score_frame['pitches
```

```
# Output Call
plt.show()
```





6.2.4. Slider Model Rankings

The following subsection ranks the pitchers who have thrown enough sliders to qualify in the 2023 season. To do this, pitchers must have thrown more than 40 sliders so far. Right away, the best expected whiff rate of all sliders is Jacob DeGrom according to the model which correlates with the public sentiment. From the simple eye test, DeGrom has an extremely effective slider and it is encouraging to see that the model valued it as such. Along with DeGrom, guys like Jordan Hicks and Jose Leclerc are know to have difficult sliders to deal with due to their velocity and rated as such by the model. It seems that the model values how hard a slider is thrown with the proper break more than if a slider is slow and loopy. There are no suprise names in the worst expected whiff rates in the league this year. It is worth noting that only two pitchers have sliders that don't have an above 0 expected whiff rate while the previous pitches all have numerous players with such. With that being said, in general it can be loosely claimed that the slider is the most effective pitch when looking to generate a swing and a miss.

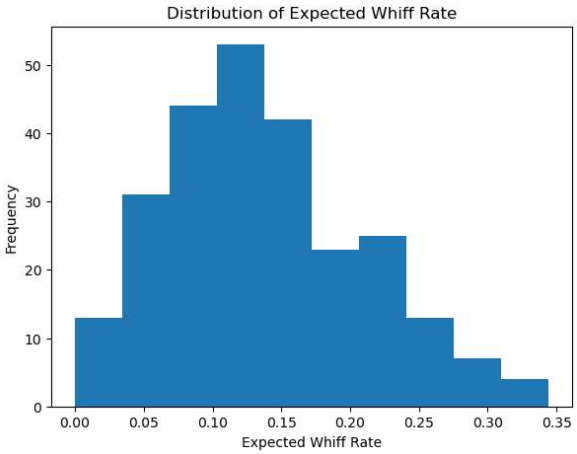
In [23]:
```
# Slider Calculations
slid_exp['expected_score'] = slid_pred
slid_pitcher_names = slid_exp['pitcher_name'].unique()
slid_score_frame = pd.DataFrame(columns = ['player_name', 'expected_score', 'pitches_c
for i in range(len(slid_pitcher_names)):
    pitcher = slid_exp[slid_exp['pitcher_name'] == slid_pitcher_names[i]]
    if pitcher.shape[0] > 40:
        values = {'player_name': slid_pitcher_names[i], 'expected_score': pitcher['exp
                                'pitches_count': pitcher.shape[0]}
        slid_score_frame = pd.concat([slid_score_frame, pd.DataFrame(values, index = |
                                      ignore_index = True)

# Rankings Output
pd.concat([slid_score_frame.sort_values(by = 'expected_score', ascending = False).iloc
           slid_score_frame.sort_values(by = 'expected_score', ascending = True).iloc|
```

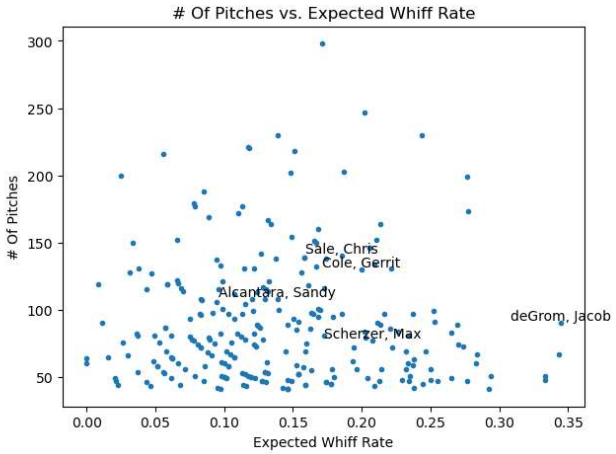
Out[23]:

	player_name	expected_score	pitches_count	player_name	expected_score	pitches_count
0	deGrom, Jacob	0.344444	90	Moll, Sam	0.000000	60
1	Faedo, Alex	0.343284	67	Cessa, Luis	0.000000	64
2	Hicks, Jordan	0.333333	48	Gilbert, Logan	0.008403	119
3	Urquidy, José	0.333333	51	Rogers, Tyler	0.011111	90
4	Medina, Luis	0.294118	51	Gomber, Austin	0.015385	65
5	Leclerc, José	0.292683	41	Cortes, Nestor	0.020408	49
6	Thompson, Mason	0.283582	67	Castillo, Diego	0.021277	47
7	King, Michael	0.283333	60	Baumann, Mike	0.022727	44
8	Gray, Jon	0.277457	173	Sears, JP	0.025000	200
9	Middleton, Keynan	0.276596	47	Kelly, Kevin	0.026316	76

Unlike the other pitch distributions, the expected whiff rate of sliders is roughly normally distributed. This seems to be due to the lack of pitchers with a 0 percent expected whiff rate. This distribution is seen clearly in the scatter plot as well. While the slider model rates DeGrom as the pitcher with the best slider in 2023, other well know pitchers such as Gerrit Cole, Chris Saale, or Sandly Alcantara are all rated as league average sliders in terms of expected whiff rates. This is interesting as the average person would rate them as better than this while the model obviously does not. Along with the curveball model, this idea will be investigated in the following section.

In [24]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(slid_score_frame['expected_score'])
sub[0].set_title("Distribution of Expected Whiff Rate")
sub[0].set_xlabel("Expected Whiff Rate")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(slid_score_frame['expected_score'], slid_score_frame['pitches_count'], '.'
sub[1].set_title("# Of Pitches vs. Expected Whiff Rate")
sub[1].set_xlabel("Expected Whiff Rate")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['deGrom, Jacob', 'Kluber, Corey', 'Scherzer, Max', 'Sale, Chris', 'Alcantar
location = np.where(np.isin(slid_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(slid_score_frame['expected_score'].iloc[loc], slid_score_frame['pitches_c

# Output Call
plt.show()
```





These types of rankings are useful as it allows for classifying effective pitches based off of historical performances of pitches like them. By identifying effective pitch designs for swings and misses pitchers can train to mirror these efficient traits. For example, we learned that for sliders the faster tighter breaking sliders are more effective than the loopy longer breaking sliders. With knowledge like this, pitchers have better guidance on where to improve. These types of rankings also allow for more effective managerial decisions as decision makers can select which pitchers they want based off these rankings for their team as well as make in game decisions on which pitches to throw depending on the situation. This swing and miss model holds a lot of potential usage for different people who make up the MLB.

6.3 Pitchers Expected vs. Actual Performance

This subsection ranks pitchers based off the difference between their actual and expected whiff rate. Unlike the previous section, now the actual performance of the pitch is being compared to what was expected. The ultimate goal of this section is to understand where the model is lacking. For example, it has been mentioned that velocity on fastballs may be undervalued by the fastball model and so be analyzing which pitchers generated swings and misses exceptionally well in comparison to the model predictions will allow for better understanding of how well the model is performing and future improvements.

6.3.1. Fastball Model Rankings

Below are the difference score rankings for the fastball model for the 2023 season with the same eligibility requirements as the expected whiff rate rankings. Examining the rankings it becomes very clear that the model is underpricing the impact of velocity on fastballs when generating swings and misses. Of the top 6 guys on the rankings, four are very well known flame throwers who all top out at over 100 mph. With their presence at the top of the rankings, it confirms prior suspicions. The bottom of the rankings also support this idea as many of the guys listed have slower velocities on their fastball and are most definetly not know for their high velocities. This is a limitation in the model and an idea that could be focused on in future research by weighting certain predictors as well. A potential reason for this lack of variance accounted for by the model may go beyond just velocity. The reason guys like Chapman, Bautista, and Alvarado are so effective is not only due to their velocity but, lack of control at

times. These big flame throwers often times lack control which can be effective against hitters as they are not sure what to expect because the guys throwing don't know what to expect. This potential phenomena may be what the model misrepesented and not the velocity component.

In [25]:
```
# Pitcher Rankings
fast_pitcher_names = fast_exp['pitcher_name'].unique()
fast_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(fast_pitcher_names)):
    pitcher = fast_exp[fast_exp['pitcher_name'] == fast_pitcher_names[i]]
    if pitcher.shape[0] > 90:
        values = {'player_name': fast_pitcher_names[i], 'score': (pitcher['expected_di
                                                                  /pitcher.shape[0]),
                                'pitches_count': pitcher.shape[0]}
        fast_score_frame = pd.concat([fast_score_frame, pd.DataFrame(values, index = |
                                     ignore_index = True)
fast_score_frame = fast_score_frame.reset_index(drop = True)

# Rankings Output
pd.concat([fast_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].r
           fast_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].re
```

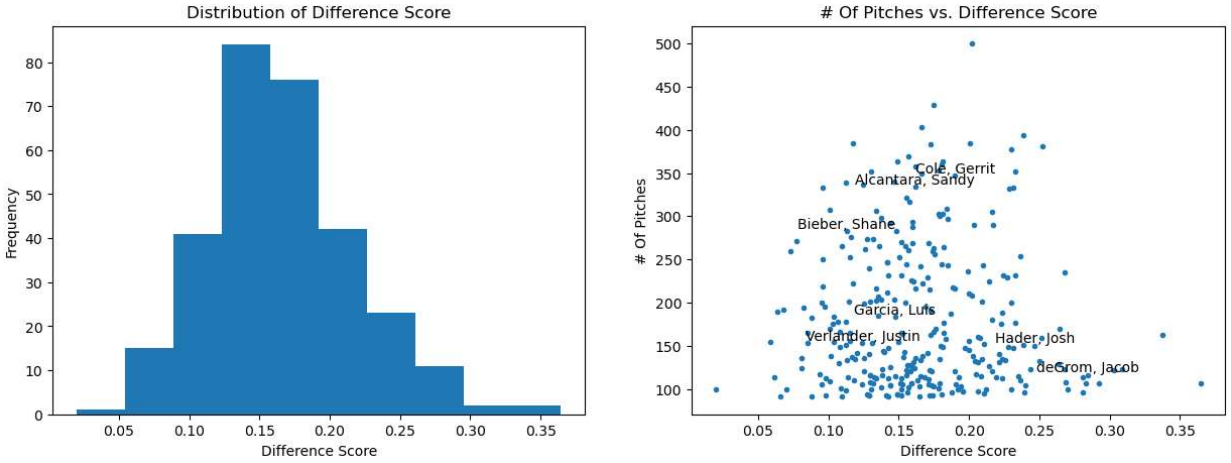
Out[25]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Chapman, Aroldis	0.364486	107	Long, Sam	0.020000	100
1	Bautista, Félix	0.337423	163	Contreras, Roansy	0.058442	154
2	Bednar, David	0.308943	123	Irvin, Jake	0.061404	114
3	Gibaut, Ian	0.303279	122	Miley, Wade	0.063492	189
4	Alvarado, José	0.292453	106	Lawrence, Justin	0.065934	91
5	deGrom, Jacob	0.284483	116	DeSclafani, Anthony	0.067708	192
6	Sewald, Paul	0.283019	106	Cessa, Luis	0.070000	100
7	Miller, Shelby	0.281250	96	Gibson, Kyle	0.073077	260
8	Kimbrel, Craig	0.280702	114	Corbin, Patrick	0.077491	271
9	Jiménez, Joe	0.270000	100	Assad, Javier	0.080645	124

The difference score distributions for the pitchers in this subsection resembles a normal distribution unlike the expected whiff rate distribution for the fastball model which were closer to a Log-normal distribution. Looking at the well known pitchers graphed on the plot, they all seem to perform around league average at generating more swing and misses than expected. This is interesting as many of these same pitchers with notoriety had very low expected whiff rates and were given opportunities to exceed expectations by a larger margin but, instead of doing so sat around league average.

In [26]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(fast_score_frame['score'])
```

```
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(fast_score_frame['score'], fast_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Cole, Gerrit', 'Bieber, Shane', 'Alcantara, Sandy', 'Verlander, Justin', '
location = np.where(np.isin(fast_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(fast_score_frame['score'].iloc[loc], fast_score_frame['pitches_count'].il

# Output Call
plt.show()
```



6.3.2. Offspeed Model Rankings

Below are the difference scores from the offspeed model from the 2023 season. These players are eligible like the previous section by throwing more than 20 offspeed pitches this season at the time of data collection. Using the player rankings generated, there are two things that we can seen. First, it seems that the model may be undervaluing the effect that hard changeups have. Secondly, it seems to value guys highly who throw circle-changes or pitches with more break on them than a straight change. This can be seen in the previous section with guys like Kevin Gausman and Max Fried being highly ranked in terms of expected whiff rate. As for the first point, looking below, Spencer Strider the top ranked player for this particular metric, throws high 90s and has a changeup that sits around 90mph. While his changeup isn't that slow or dropping a ton like other pitchers, his changeup is effective because it is slow relative to his fastball. This extends beyond just Strider as other guys on the list have a similar changeup style to generate misses. The model has no chance to capture this relationship because the attributes of other pitches weren't considered when fitting the offspeed model. This is a limitation however, provides insight on where to grow this model.

In [27]:
```
# Offspeed Calculations
off_pitcher_names = off_exp['pitcher_name'].unique()
off_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(off_pitcher_names)):
    pitcher = off_exp[off_exp['pitcher_name'] == off_pitcher_names[i]]
    if pitcher.shape[0] > 20:
        values = {'player_name': off_pitcher_names[i], 'score': (pitcher['expected_dif
                                                                 /pitcher.shape[0]),
                            'pitches_count': pitcher.shape[0]}
        off_score_frame = pd.concat([off_score_frame, pd.DataFrame(values, index = [0
                                     ignore_index = True)

# Rankings Output
pd.concat([off_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].re
           off_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].res
```

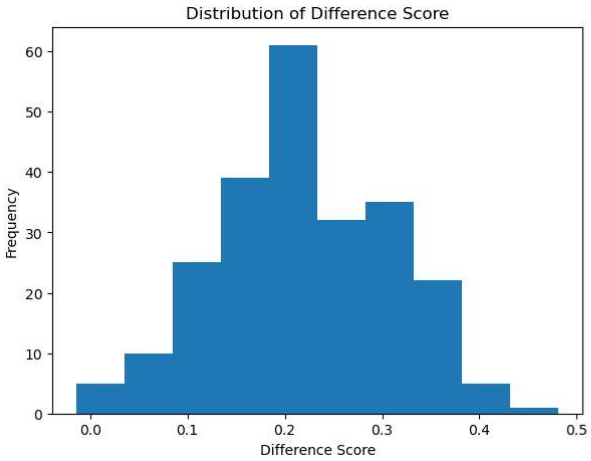
Out[27]:

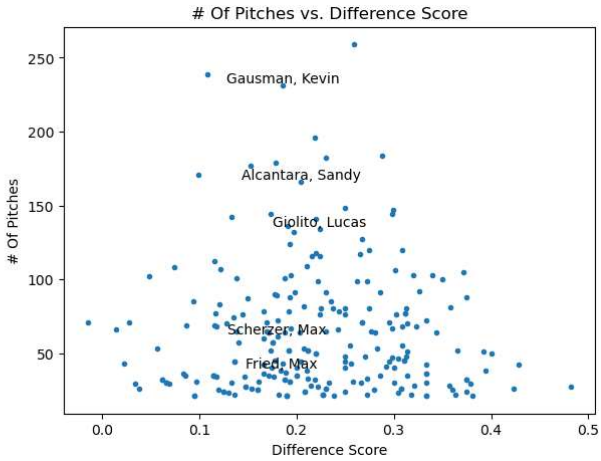
	player_name	score	pitches_count	player_name	score	pitches_count
0	Strider, Spencer	0.481481	27	Greinke, Zack	-0.014085	71
1	Springs, Jeffrey	0.428571	42	Miley, Wade	0.015152	66
2	Lynch, Daniel	0.423077	26	DeSclafani, Anthony	0.023256	43
3	Houck, Tanner	0.400000	50	Quantrill, Cal	0.028169	71
4	Barria, Jaime	0.394737	38	Alexander, Tyler	0.034483	29
5	Bibee, Tanner	0.392157	51	Bradish, Kyle	0.038462	26
6	Lee, Dylan	0.380952	21	Syndergaard, Noah	0.049020	102
7	Leclerc, José	0.379310	29	Freeland, Kyle	0.056604	53
8	Pfaadt, Brandon	0.375000	24	Mikolas, Miles	0.062500	32
9	Weaver, Luke	0.375000	88	Faedo, Alex	0.062500	32

The distribution of these scores are once again normal, which can be seen below. As for the scatter plot, there are no noticeable points that stand out. All the well known pitchers who have an offspeed pitch are right in the middle of the league in terms of difference score. This is impressive as guys like Kevin Gausman and Max Fried were highly rated by the model and still had a higher whiff rate on their changeup than the model predicted.

In [28]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(off_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(off_score_frame['score'], off_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")
```

```
### Adding Popular Player Names

players = ['Gausman, Kevin', 'Giolito, Lucas', 'Alcantara, Sandy', 'Fried, Max', 'Sche
location = np.where(np.isin(off_score_frame['player_name'], players))
for loc in location[0]:
    plt.text(off_score_frame['score'].iloc[loc], off_score_frame['pitches_count'].iloc

# Output Call
plt.show()
```





6.3.3. Curveball Model Rankings

This next subsection looks at the difference score for the curveball model for the 2023 season. This model has the same requirements of eligibility as the previous subsection, more than 10 curveballs. Looking at the pitchers who exceeded expectations, there are a few who excelled extremely well. With that being said, there is no particular curveball type that seems to be undervalued. Eury Perez and Jesse Chavez both have very different curveball types however, both excelled greatly in comparison to what the model predicted. There does not seem to be a specific reason for the lack of variance accounted for in the model for some of these players. This in it of itself is a limitation as it is not apparrent where to grow the accurracy and when the model may make faulty predictions.

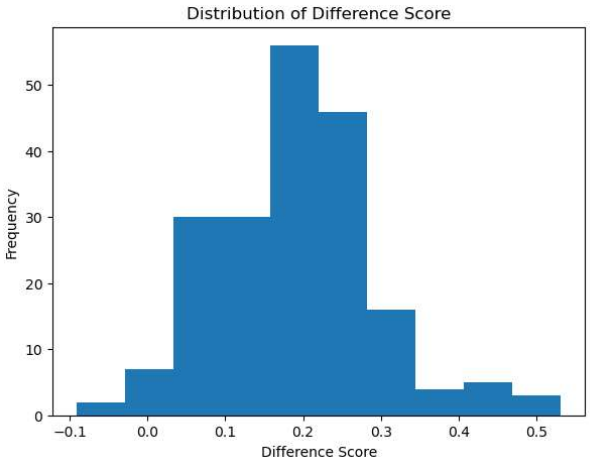
In [29]:
```
# Curveall Calculations
curve_pitcher_names = curve_exp['pitcher_name'].unique()
curve_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(curve_pitcher_names)):
    pitcher = curve_exp[curve_exp['pitcher_name'] == curve_pitcher_names[i]]
    if pitcher.shape[0] > 10:
        values = {'player_name': curve_pitcher_names[i], 'score': (pitcher['expected_
                                                                   /pitcher.shape[0]),
                                  'pitches_count': pitcher.shape[0]}
        curve_score_frame = pd.concat([curve_score_frame, pd.DataFrame(values, index =
                                       ignore_index = True)

# Rankings Output
pd.concat([curve_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,],
           curve_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].r
```

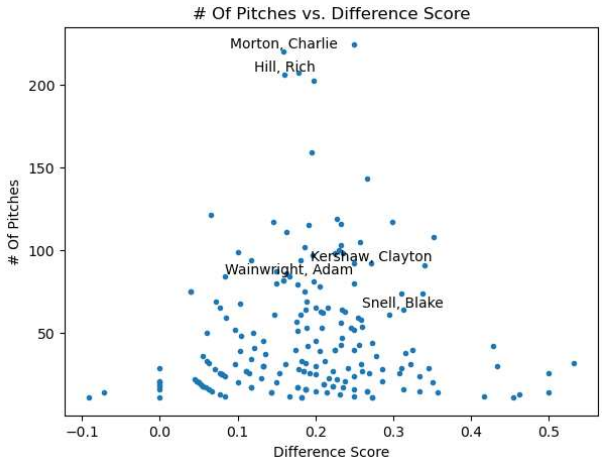
Out[29]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Pérez, Eury	0.531250	32	Ramírez, Erasmo	-0.090909	11
1	France, J.P.	0.500000	26	Misiewicz, Anthony	-0.071429	14
2	Chavez, Jesse	0.500000	14	Hernández, Carlos	0.000000	18
3	Smith, Will	0.461538	13	Moore, Matt	0.000000	21
4	Thielbar, Caleb	0.454545	11	Sands, Cole	0.000000	11
5	Bush, Matt	0.433333	30	Mantiply, Joe	0.000000	16
6	Darvish, Yu	0.428571	42	Gray, Jon	0.000000	19
7	Leiter Jr., Mark	0.416667	12	Hunter, Tommy	0.000000	20
8	Lambert, Jimmy	0.357143	14	Garrett, Braxton	0.000000	29
9	Gore, MacKenzie	0.351852	108	Matz, Steven	0.040000	75

The distribution of the difference scores from the curveball model for the 2023 season seem to follow a normal distribution roughly. When examining the scatter plot containing some well known curveball pitchers, nothing jumps off the page. There were concerns in the previous section about Clayton Kershaw being undervalued but, he doesn't seem to be undervalued more than the average MLB curveball pitcher. Looking at the plot, he is around average at exceeding the expected whiff rate generated by the model. This sentiment doesn't mean Kershaw is accounted for perfectly however, the concerns mentioned in the previous curveball subsection are somewhat mitigated.

In [30]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(curve_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(curve_score_frame['score'], curve_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['Kershaw, Clayton', 'Wainwright, Adam', 'Morton, Charlie', 'Hill, Rich', 'S
location = np.where(np.isin(curve_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(curve_score_frame['score'].iloc[loc], curve_score_frame['pitches_count'],

# Output Call
plt.show()
```





6.3.4. Slider Model Rankings

This next subsection discusses the difference scores generated from the slider model from the 2023 season. To be eligibile, the pitchers had to throw 40 or more sliders thus far into the season. Like the curveball model, there doesn't seem to be an apparent type of slider that is undervalued in the model. The top guys listed don't have a distinct slider type and therefore their effectiveness may be due to the effect of their other pitches and how they play off the slider. One interesting note in the worst difference scores of eligible pitchers is Shane Bieber. He used to be regarded as one of the top pitchers in the league however, has fallen off recently. This performance is reflected by the model and how his pitches generate swings and misses as he failed to beat expectations similarly to other MLB pitchers so far this year.

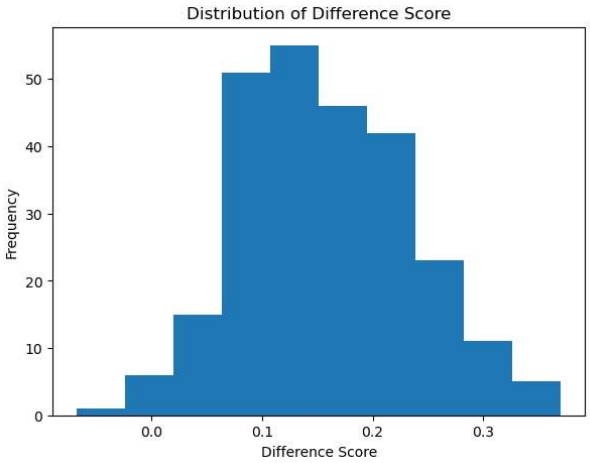
In [31]:
```
# Slider Calculations
slid_pitcher_names = slid_exp['pitcher_name'].unique()
slid_score_frame = pd.DataFrame(columns = ['player_name', 'score', 'pitches_count'])
for i in range(len(slid_pitcher_names)):
    pitcher = slid_exp[slid_exp['pitcher_name'] == slid_pitcher_names[i]]
    if pitcher.shape[0] > 40:
        values = {'player_name': slid_pitcher_names[i], 'score': (pitcher['expected_di
                                                                  /pitcher.shape[0]),
                            'pitches_count': pitcher.shape[0]}
        slid_score_frame = pd.concat([slid_score_frame, pd.DataFrame(values, index = |
                                ignore_index = True)

# Rankings Output
pd.concat([slid_score_frame.sort_values(by = 'score', ascending = False).iloc[0:10,].r
           slid_score_frame.sort_values(by = 'score', ascending = True).iloc[0:10,].re
```

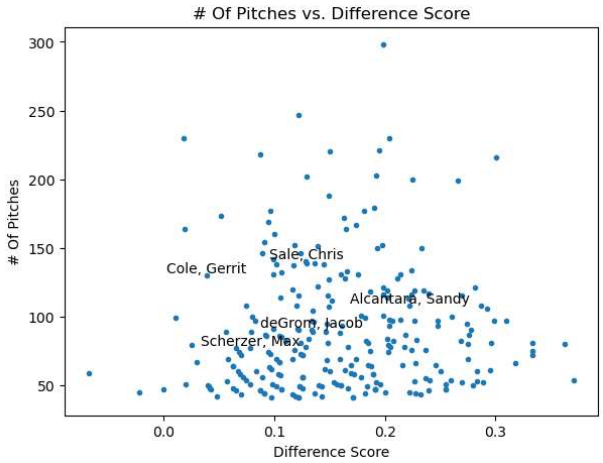
Out[31]:

	player_name	score	pitches_count	player_name	score	pitches_count
0	Adam, Jason	0.370370	54	Dunning, Dane	-0.067797	59
1	Cabrera, Génesis	0.362500	80	Pruitt, Austin	-0.022222	45
2	Helsley, Ryan	0.333333	75	Williams, Trevor	0.000000	47
3	Chafin, Andrew	0.333333	72	Heaney, Andrew	0.010101	99
4	Bassitt, Chris	0.333333	81	Singer, Brady	0.017391	230
5	Holmes, Clay	0.318182	66	Bieber, Shane	0.018293	164
6	Blanco, Ronel	0.309278	97	Urquidy, José	0.019608	51
7	Detmers, Reid	0.300926	216	Barlow, Scott	0.025316	79
8	Gallegos, Giovanny	0.298969	97	Thompson, Mason	0.029851	67
9	Soto, Gregory	0.296296	81	Cole, Gerrit	0.038462	130

Like the other difference score distributions discussed in this subsection, the distribution from the slider model follows roughly a normal distribution. Looking at the popular names listed on the slider model plot, there aren't any interesting suprises. The more well know players performed worse than league average in comparison to their own expected whiff rates however, for the slider model a lot of the "better" players were represented similarly to their notoriety. With that being said, their performance compared to other players is still better and they just didn't outperform their already large expectations as much.

In [32]:
```
# Graphical Output
## Configuring Size
figure, sub = plt.subplots(1,2, figsize = (15,5))

## Plot 1
sub[0].hist(slid_score_frame['score'])
sub[0].set_title("Distribution of Difference Score")
sub[0].set_xlabel("Difference Score")
sub[0].set_ylabel("Frequency")

# Plot 2
sub[1].plot(slid_score_frame['score'], slid_score_frame['pitches_count'], '.')
sub[1].set_title("# Of Pitches vs. Difference Score")
sub[1].set_xlabel("Difference Score")
sub[1].set_ylabel("# Of Pitches")

### Adding Popular Player Names
players = ['deGrom, Jacob', 'Kluber, Corey', 'Scherzer, Max', 'Sale, Chris', 'Alcantar
location = np.where(np.isin(slid_score_frame['player_name'], players))

for loc in location[0]:
    plt.text(slid_score_frame['score'].iloc[loc], slid_score_frame['pitches_count'].il

# Output Call
plt.show()
```





These difference scores are effective to analyze as they allow for more insight on the performance of the model. Not only can we see how well it predicted whiff rate on certain pitches but, by examining who had the largest discrepancies we can learn what type of pitch designs aren't being represented as accurately in the model. By learning these limitations decisions can be made more insightfuly as well as we can learn what to look to improve in the future. Lastly, almost all the difference scores were positive and the reason for this was the higher amount of contact than misses in the training as well as testing data. For this reason, there are much more false contact classifications than false miss classifications. This is because most of the expected whiff rates to be lower than the true whiff rate for the particular pitch. The expected whiff rates in the second subsection were more to rank players and their relative ability than to predict their actual expected whiff rate. It is also worth noting that whiff rates were higher in the 2023 testing data for every pitch type across the board than the 2022 training data, which may contribute the underestimated whiff rates.

7. Performance Difference Case Study - Sandy Alcantara

In this section, we will be using the generated pitch models in order to conduct a case study on a specific player's performance, Sandy Alcantara. Alcantara won the Cy Young Award for the National League in 2022 due to having a dominant year. With that being said, Alcantara has had a completely different year so far in 2023. The below figure shows a comparison of some of Alcantara's stats in both the year 2022 and 2023. His stats have fallen off in every category shown below from 2022 to 2023. There was speculation about the year Alcantara was going to have from many analysts who saw his performance drop in the 2023 WBC and some even forecasted him having this type of year. Some believe he is fatigued from 2022 as he worked through many innings that year and routinely went far into games while others believe he is battling some sort of injury. Regardless of the reason, this section will be looking into whether a noticeable difference can be spotted between the two years and the expected whiff rate from Alcantara's pitches. Alcantara does not throw a curveball and so ony the fastball, offspeed, and slider model will be used for this study.

In [45]:
```
# 2022 Expected Values
## Creating Result Predictions For 2022 Pitches
fast_pred_2022 = fast_tree.predict(fast_data)
```

```
off_pred_2022 = off_tree.predict(off_data)
curve_pred_2022 = curve_tree.predict(curve_data)
slid_pred_2022 = slid_tree.predict(slid_data)

## Pitcher Expected Values
fast_exp_2022['expected_score'] = fast_pred_2022
off_exp_2022['expected_score'] = off_pred_2022
curve_exp_2022['expected_score'] = curve_pred_2022
slid_exp_2022['expected_score'] = slid_pred_2022

## Pitcher Difference Values
fast_exp_2022['expected_difference_score'] = fast_miss - fast_pred_2022
off_exp_2022['expected_difference_score'] = off_miss - off_pred_2022
curve_exp_2022['expected_difference_score'] = curve_miss - curve_pred_2022
slid_exp_2022['expected_difference_score'] = slid_miss - slid_pred_2022

# Adding Miss Values
## 2022
fast_exp_2022['miss'] = fast_miss
off_exp_2022['miss'] = off_miss
curve_exp_2022['miss'] = curve_miss
slid_exp_2022['miss'] = slid_miss

## 2023
fast_exp['miss'] = fast_miss_test
off_exp['miss'] = off_miss_test
curve_exp['miss'] = curve_miss_test
slid_exp['miss'] = slid_miss_test

# Loading In Sandy Stats
sandy = pd.read_csv('C:/Users/tscot/Downloads/PersonalProjects/swingmissproj/sandydata
sandy = sandy[sandy['last_name'] == "Alcantara"]

# Year Comparison
year_comp = sandy.iloc[:,[3, 4, 6, 8, 9, 12, 20]]
print(tabulate(year_comp, headers = year_comp.columns, tablefmt = "fancy_grid"))
```

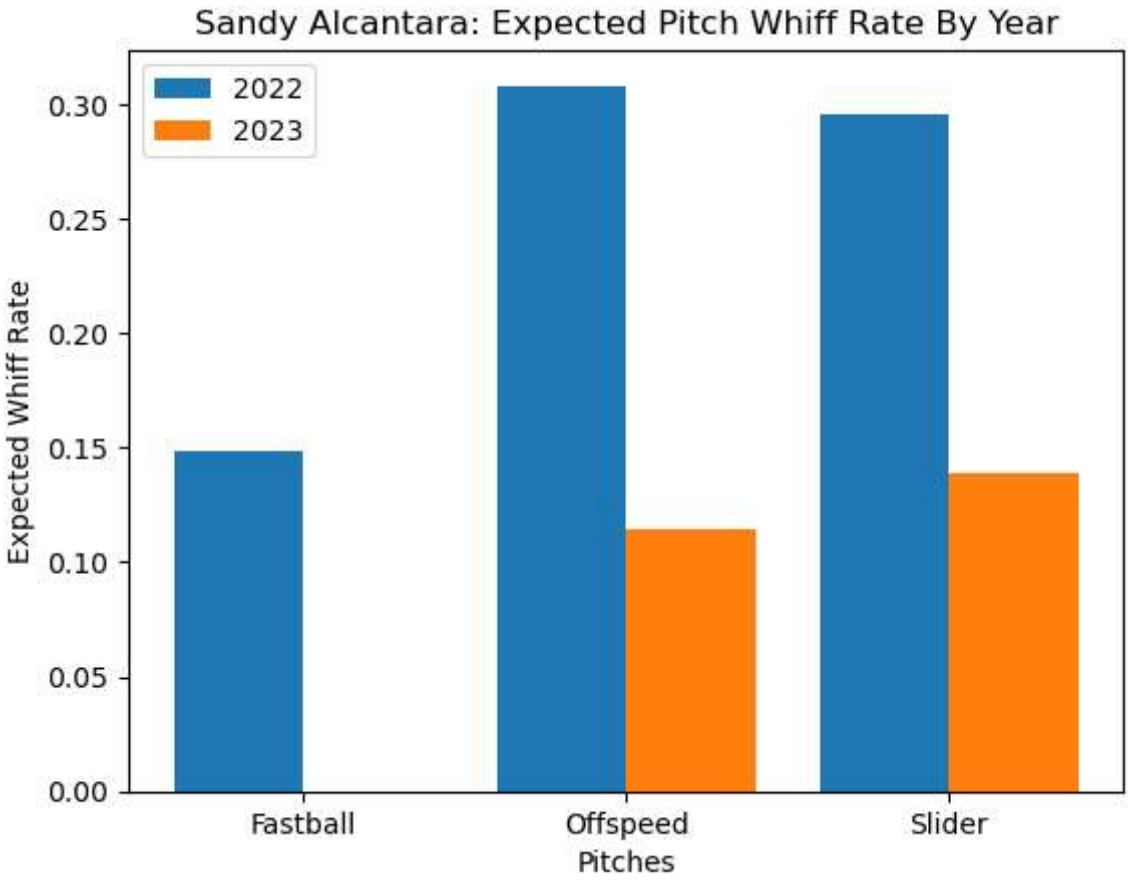
```
╒═══════╤════════╤═══════════════╤═══════════════════╤═══════════╤═══════╤═══════════════
═══╤═══════════════╕
│       │   year │   batting_avg │   on_base_percent │   p_era │   xba │   barrel_batte
d_rate │   whiff_percent │
╞═══════╪════════╪═══════════════╪═══════════════════╪═══════════╪═══════╪═══════════════
═══╪═══════════════╡
│  42 │   2022 │         0.212 │             0.263 │    2.28 │ 0.215 │
5.3 │            25.4 │
├───────┼────────┼───────────────┼───────────────────┼───────────┼───────┼───────────────
───┼───────────────┤
│ 207 │   2023 │         0.231 │             0.297 │    4.75 │ 0.254 │
6.2 │            26.6 │
╘═══════╧════════╧═══════════════╧═══════════════════╧═══════════╧═══════╧═══════════════
═══╧═══════════════╛
```

First, we will look at the expected whiff rates over the two years on Alcantara's pitches. Despite his whiff rate being better so far in 2023 than 2022, it will be interesting to see if the model projects any differences between the pitches. While the actual whiff rates may not differ too much, if the expected whiff rates are different it could be a sign of worse performing pitches that are getting hit harder. Looking at the plot below, there is a very noticeable difference between Alcantara's expected whiff rates between the two years. Across the board, every pitch

is expected to perform better in the context of generating swings and misses in 2022 than 2023. The offspeed and slider are almost three times as high in 2022 than 2023. These drastic differences signify that there could be a problem in Alcantara's pitch design and the types of pitches he are throwing are not as effective. This could be due to any adjustments he has made as well as a fatigue issue. What is interesting is that even though the model has identified differences between his pitches from 2022 to 2023, the actual whiff rates have not differed all too much. This points to a batted ball issue as it seems that generating misses aren't an issue but, the type of contact is what is causing the problems. The model identifies misses by classifying difficult pitches to hit as pontential misses. Due to the lack of expected misses, this means that the difficulty of the pitches has decreased therefore, on contact, these pitches are more likely to have explosive results. This can be seen in the above figure showing Alcantara's barrel rate which has jumped from 2022 to 2023.

In [46]:
```
# Sandy Average Pitches

## 2022
sandy_fast_2022 = fast_exp_2022[fast_exp_2022['pitcher_name'] == "Alcantara, Sandy"]
sandy_off_2022 = off_exp_2022[off_exp_2022['pitcher_name'] == "Alcantara, Sandy"]
sandy_slid_2022 = slid_exp_2022[slid_exp_2022['pitcher_name'] == "Alcantara, Sandy"]


## 2023
sandy_fast_2023 = fast_exp[fast_exp['pitcher_name'] == "Alcantara, Sandy"]
sandy_off_2023 = off_exp[off_exp['pitcher_name'] == "Alcantara, Sandy"]
sandy_slid_2023 = slid_exp[slid_exp['pitcher_name'] == "Alcantara, Sandy"]

## Collecting Average Expected Scores
sandy_list = [sandy_fast_2022, sandy_off_2022, sandy_slid_2022, sandy_fast_2023, sandy
sandy_values = []
for i in range(6):
    sandy_values.append(sandy_list[i]['expected_score'].sum()/sandy_list[i].shape[0])

# Output
## Plot Code
pitches = ['Fastball', 'Offspeed', 'Slider']
years = ['2022', '2023']

# Set the positions of the bars on the x-axis
pos = np.arange(len(pitches))

# Create the figure and axes
fig, ax = plt.subplots()

# Plot the bars for each group
bar1 = ax.bar(pos - 0.2, sandy_values[0:3], width = 0.4, label=years[0])
bar2 = ax.bar(pos + 0.2, sandy_values[3:], width = 0.4, label=years[1])

# Add labels, title, and legend
ax.set_xlabel('Pitches')
ax.set_ylabel('Expected Whiff Rate')
ax.set_title('Sandy Alcantara: Expected Pitch Whiff Rate By Year')
ax.set_xticks(pos)
ax.set_xticklabels(pitches)
ax.legend()
```

```
# Show the plot
plt.show()
```



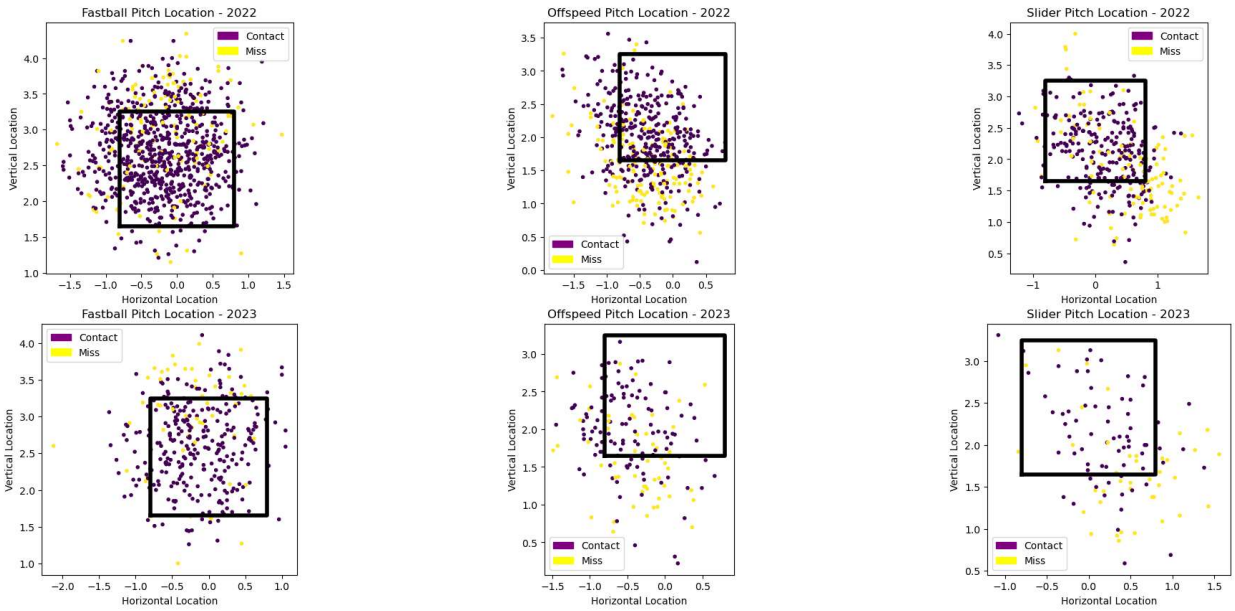
Despite not being able to judge the contact with the current state of the models, there are still visualizations that can give help give insight. Below are six plots showing the location of each pitch type between the two years along with whether that pitch was a miss or contact. The idea is to see if the plots give a visual representation on what is causing Sandy to give up harder contact. Examining the locations of each of the pitches it is not particularly evident. Looking closely at the plots of the sliders, it seems that Sandy is not locating as well in 2023 as 2022. Ideally sliders will finish slightly off the edge of the plate against a righty batter making them appear to be a strike and break hard away. The 2022 slider plot is filled with located pitches just off the edge while in 2023 he doesn't hit that edge as consistently. This has a potential to explain why the contact has been harder as Sandy's wipeout pitch isn't being as located as effectively. With that being said, this is a rough estimate from the plots and no particular trend stands out exclusively.

In [47]:
```
import matplotlib.patches as mpatches

# Pitch Location On Swings

## Figure Configuration
figure, sub = plt.subplots(2,3, figsize = (24,10))

## 2022
### Fastball
sub[0,0].scatter(sandy_fast_2022['plate_x'], sandy_fast_2022['plate_z'], marker = '.',
```

```
sub[0,0].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[0,0].set_title("Fastball Pitch Location - 2022")
sub[0,0].set_xlabel("Horizontal Location")
sub[0,0].set_ylabel("Vertical Location")
sub[0,0].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[0,0].legend(handles = [contact, miss])


### Offspeed
sub[0,1].scatter(sandy_off_2022['plate_x'], sandy_off_2022['plate_z'], marker = '.', c
sub[0,1].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[0,1].set_title("Offspeed Pitch Location - 2022")
sub[0,1].set_xlabel("Horizontal Location")
sub[0,1].set_ylabel("Vertical Location")
sub[0,1].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[0,1].legend(handles = [contact, miss])

### Slider
sub[0,2].scatter(sandy_slid_2022['plate_x'], sandy_slid_2022['plate_z'], marker = '.',
sub[0,2].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[0,2].set_title("Slider Pitch Location - 2022")
sub[0,2].set_xlabel("Horizontal Location")
sub[0,2].set_ylabel("Vertical Location")
sub[0,2].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[0,2].legend(handles = [contact, miss])

## 2023
### Fastball
sub[1,0].scatter(sandy_fast_2023['plate_x'], sandy_fast_2023['plate_z'], marker ='.',
sub[1,0].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[1,0].set_title("Fastball Pitch Location - 2023")
sub[1,0].set_xlabel("Horizontal Location")
sub[1,0].set_ylabel("Vertical Location")
sub[1,0].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[1,0].legend(handles = [contact, miss])

### Offspeed
sub[1,1].scatter(sandy_off_2023['plate_x'], sandy_off_2023['plate_z'], marker = '.', c
sub[1,1].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[1,1].set_title("Offspeed Pitch Location - 2023")
sub[1,1].set_xlabel("Horizontal Location")
sub[1,1].set_ylabel("Vertical Location")
sub[1,1].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[1,1].legend(handles = [contact, miss])

### Slider
sub[1,2].scatter(sandy_slid_2023['plate_x'], sandy_slid_2023['plate_z'], marker = '.',
sub[1,2].plot([-0.8, 0.8, 0.8, -0.8, -0.8], [1.65, 1.65, 3.25, 3.25, 1.65], linewidth
sub[1,2].set_title("Slider Pitch Location - 2023")
sub[1,2].set_xlabel("Horizontal Location")
```

```
sub[1,2].set_ylabel("Vertical Location")
sub[1,2].set_aspect("equal")
contact = mpatches.Patch(color = "purple", label = "Contact")
miss = mpatches.Patch(color = "yellow", label = "Miss")
sub[1,2].legend(handles = [contact, miss])
```

Out[47]: <matplotlib.legend.Legend at 0x2ad8d8d0fa0>



All in all, there has been a huge difference between Alcantara's 2022 and 2023 performances. Using the model, it was able to be identified that there is something off about the pitches he is throwing because the drop off of expected whiff rate across all pitch categories Alcantara qualified for. Despite his actual whiff rate increasing, the significantly lower expected whiff rates signify a lack of effectiveness in his pitches that are most likely resulting in harder contact. The fix to this issue isn't able to be presented by the models however, by using them Alcantara's lack of effectiveness was able to be identified and not subjected to outcome variance. In the future, this type of swing and miss model can be used to quantify the difference in pitcher performance over the years to diagnose if there is a significant difference in ability or, if the different levels of performance are due to variance.

8. Conclusion

In the future, the current model could be extended in multiple ways. First, identifying a way to better capture all the variance for each pitch type would be a good practice. It was identified in the fastball and offspeed models that certain types of those pitches may not have been represented entirely. By doing so, the accuracy can be increased when predicting misses while properly valuing all types of pitchers. Secondly, building the model to incorporate the way different pitch types interact could lead to a higher accuracy when predicting misses. A pitcher's pitches are effective at times with the way they play off of each other. A 90mph changeup isn't effective when the pitcher throws a 93mph fastball but, when he throws 100mph that changeup now become effective. The way the pitches play off each other matter and by incorporating this through some sort of interaction effect could be an effective approach to build this idea into

the model. Lastly, a future research point could be to attempt to fit models using different algorithms. While a basic feedforward neural network was attempted and not as effective as a random forest, other types of neural networks such as a recurrent neural network or other types of algorithms could prove to be even more effective when predicting misses.

In conclusion, this research was able to fit four different accurate pitch models to significantly predict whether a given pitch will result in a miss when swung at based off the metrics of that pitch. With these fitted models, hitters were able to be fairly ranked based off contact ability and pitchers were able to be fairly ranked based off their ability to generated expected misses. As mentioned in the introduction, by using this type of model to formulate rankings contact abilities can be rated as they become increasingly more important with the new rule changes. Along with the ability to rank player performance and ability, the model presents an opportunity to identify differences in ability of players over different seasons. The pitch models can be effectively utilized to aid with pitch design and help pitchers identify the optimal pitch types to generate swings and misses. Lastly, managers can use the generated models to make in game decisions when needing to call a pitch to generate a swing and a miss in a crutial situation. This model can help determine what pitch to call and where to throw it based off what is most likely to lead to a miss.