

✓ Machine Learning to Explore Statcast Data from the 2024 MLB Season

Nolan Lo

UCSD Data Science Masters Student

December 10th, 2024

The purpose of this project is purely for research and exploration purposes. The questions being asked and explored are not picked with the intent to optimize model accuracy, but rather using machine learning to explore topics of interest to inquire if these topics are worth optimizing and investing into. The 3 questions that will be explored are below:

✓ Research Questions

1) Pitch Analysis: Can pitch characteristics such as pitch type, velocity, spin rate, pitch movement, and pitch location, predict the probability of a swing-and-miss (whiff rate)?

2) Batter Talent Can we use features that are inherently "pure raw talent" to predict a batter's success? "Pure raw talent" would equate to traits that a batter is born with, similar to how basketball coaches will say that you can't teach height. For a batter, I would classify this as their bat speed and swing length. In my opinion, these are traits that some batters are naturally much better at than others as a result of natural born talent. Other traits can be trained such as pitch recognition, not chasing pitches, barreling up a ball, and launch angle. The ability to swing the bat faster and with a more efficient bat path at a much higher level than the average professional is something that I do not think can be trained, but rather a god gifted talent.

3) Situational Pitching Strategy How do pitch selection and pitch effectiveness (measured by opposing batter wOBA) vary by inning and game state (outs, inning, score difference, and base runners) on pitches that were hit? This is a two part question. First, on pitches that resulted in an out, can pitch selection be predicted based on outs, inning, score difference, and base runners? Second, with those different game states, which pitches are most effective based on estimated wOBA.

```
#Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Getting Data set
from pybaseball import statcast
df_raw = statcast(start_dt="2024-03-28", end_dt="2024-09-30")
```

Explanation of Data Set:

This data set contains information of every single pitch of every single game in the MLB 2024 season. Each row represents data from 1 pitch. Some of the fields provided in this data set include pitcher name, date, home team, away team, batter id, pitch result, pitch speed, pitch location, etc.

✓ 1) Pitch Analysis

Can pitch characteristics such as pitch type, velocity, spin rate, pitch movement, and pitch location, predict the probability of a swing-and-miss (whiff rate)?

Choice of model: Logistic regression model would work well for this question since the target variable only has two outcomes, swing-and-miss or no swing-and-miss.

```
# Set up df
df_pitch_analysis = df[['pitch_name', 'release_speed', 'release_spin_rate', 'pfx_x', 'pfx_z', 'plate_x', 'plate_z']
print(df_pitch_analysis['description'].unique())

↵ ['hit_into_play' 'ball' 'swinging_strike' 'blocked_ball' 'foul'
   'called_strike' 'swinging_strike_blocked' 'hit_by_pitch' 'foul_tip'
   'foul_bunt' 'pitchout' 'missed_bunt' 'bunt_foul_tip']

# Remove unnecessary rows. We only need pitches that the batter swung at.
df_pitch_analysis = df_pitch_analysis[df_pitch_analysis['description'].isin(['hit_into_play', 'swinging_strike',
print(df_pitch_analysis['description'].unique())

↵ ['hit_into_play' 'swinging_strike' 'foul' 'swinging_strike_blocked'
   'foul_tip']

# Keep only the pitches I want to include in this study (the common pitches)
df_pitch_analysis = df_pitch_analysis[df_pitch_analysis['pitch_name'].isin(['4-Seam Fastball', 'Slider', 'Curveball',
'Sinker', 'Changeup', 'Split-Finger'])]

# Create feature variable column
df_pitch_analysis['whiff'] = df_pitch_analysis['description'].apply(lambda x: 1 if x in ['swinging_strike', 'swi

# Encode pitch_type
df_pitch_analysis = pd.get_dummies(df_pitch_analysis, columns=['pitch_name'], drop_first=True)

# Split data set into features and target
X = df_pitch_analysis.drop(columns=['description', 'whiff'])
y = df_pitch_analysis['whiff']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 562)

# SCALE THE DATA
# Import the necessary library
from sklearn.preprocessing import StandardScaler

# Create the scaling function
scaler = StandardScaler().fit(X_train)

# Apply the scaling function to the features (0 mean and unit variance)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```



```
LogisticRegression ⓘ ?
LogisticRegression()
```

```
# Predict the labels for the data
y_predict = model.predict(X_test)

# Calculate the score (the average rate of correct classifications on a bunch of data)
score = model.score(X_test, y_test)
print(score)
```

0.7745859370260268

According to the accuracy score of 0.7745, this model's accuracy score is correct about 77.46% of the time. This means that 77% of our guesses are correct for if the batter will whiff or not based on the selected features.

```
# Predict the probabilities of each label for the data

# Note that the sum of the two columns is always 1
probs = model.predict_proba(X_test)
probs
```

array([[0.62250562, 0.37749438],
[0.79785387, 0.20214613],
[0.74955664, 0.25044336],
...,
[0.77407811, 0.22592189],
[0.90544365, 0.09455635],
[0.78445647, 0.21554353]])

This array shows each pitch, and their probability to not swing-and-miss and to swing-and-miss, respectively.

```
# CROSS VALIDATION
```

```
scores = []
```

```
for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i)

    # Scale the data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train the model
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # Evaluate the model
    score = model.score(X_test, y_test)
    scores.append(score)
```

```
# Calculate the average accuracy
average_accuracy = np.mean(scores)
print("Average accuracy:", average_accuracy)
```

Average accuracy: 0.7750970697081841

What I did in the code above was run the model 10 different times and take the average of each of the accuracy scores. This accounts for potential randomness that could have swayed our initial 1 run of the model. Seeing as the average of the 10 scores we got was still roughly 77.5%, it seems reasonable that this is a reliable accuracy score. If we were to use a baseline of 70% as a good accuracy score, that would mean this logistic model with the features we picked is a good predictor of whiffs.

✓ 2) Batter Talent

Can we use features that are inherently "pure raw talent" to predict a batter's success? "Pure raw talent" would equate to traits that a batter is born with, similar to how basketball coaches will say that you can't teach height. For a batter, I would classify this as their bat speed and swing length. In my opinion, these are traits that some batters are naturally much better at than others as a result of natural born talent. Other traits can be trained such as pitch recognition, not chasing pitches, barreling up a ball, and launch angle. The ability to swing the bat faster and with a more efficient bat path at a much higher level than the average professional is something that I do not think can be trained, but rather a god gifted talent.

The target variable I will be picking is OPS. Very generally, the higher the OPS, the better the batter. In order to see if we can use the raw talent features to predict batter success (OPS), we will have to first see if there is any relationship. We can accomplish this first by joining the two data sets, and graphing them with a scatter plot.

```
# Importing 2nd data set with batter name and stats (target variable)
from pybaseball import batting_stats_bref
batter_data = batting_stats_bref(2024)[['mlbID', 'OPS']]
batter_data.head(1)
```

	mlbID	OPS
1	682928	0.747

```
# Set up df with necessary cols
df_batter_talent = df[['batter', 'bat_speed', 'swing_length']].dropna()
```

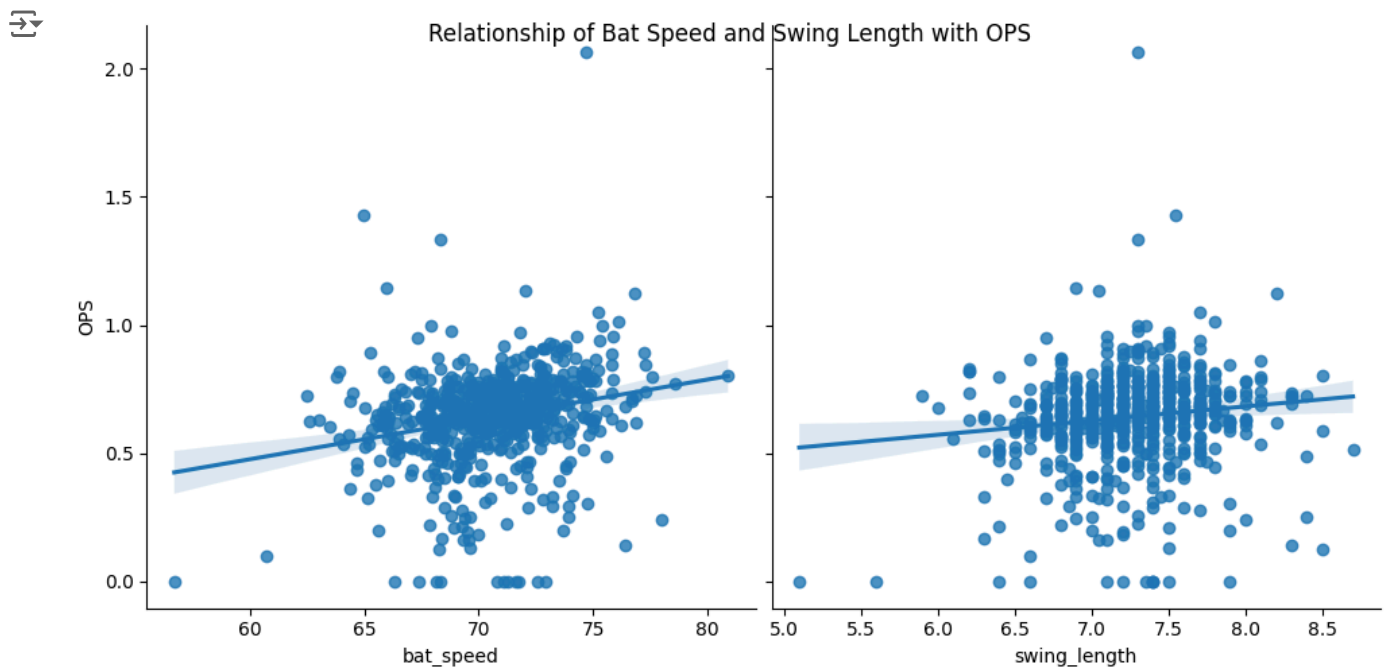
```
# Calculate median bat speed and swing length
df_batter_talent = df_batter_talent.groupby('batter').agg({
    'bat_speed': 'median',
    'swing_length': 'median'
}).reset_index()
```

```
df_batter_talent = df_batter_talent.merge(batter_data, left_on='batter', right_on='mlbID', how='inner').drop('col', axis=1)
df_batter_talent.head(1)
```

	bat_speed	swing_length	OPS
0	71.0	7.1	0.759

Now that we have our combined data set with the features and target variable, we can visualize the relationship.

```
sns.pairplot(df_batter_talent, x_vars = ['bat_speed', 'swing_length'], y_vars = 'OPS', height = 5, aspect = 1, kind='scatter')
plt.suptitle("Relationship of Bat Speed and Swing Length with OPS")
plt.show()
```



The two features do seem to have enough of a linear relationship to move on to our model of choice, linear regression. Since the target variable is numerical and has more than 2 values, linear regression fits the criteria.

```
X = df_batter_talent[['bat_speed', 'swing_length']]
y = df_batter_talent['OPS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 562)

Lin_reg = LinearRegression()
Lin_reg.fit(X_train, y_train)

print("Coefficients ->", Lin_reg.coef_)
print("Intercept ->", Lin_reg.intercept_)

Coefficients -> [0.0140104  0.00719334]
Intercept -> -0.4066199943913301
```

Both coefficients are positive, so OPS increases as the features increase. However, they are both pretty small so the increment of increase will be low.

```
# Apply to test data
y_pred_test = Lin_reg.predict(X_test)
print("Mean Squared Error ->", mean_squared_error(y_test, y_pred_test))
print("R^2 ->", r2_score(y_test, y_pred_test))

Mean Squared Error -> 0.04426229393062642
R^2 -> 0.02303097520848907
```

The MSE of 0.044 suggests that there is a small difference between the predicted OPS and actual OPS values. However, this is misleading since the R^2 value of 0.023 shows us that only 2.3% of the variance in OPS is explained by bat speed and swing length. That is extremely low, so that indicates that the model is not effective at all. In conclusion, we cannot predict batter success with just features that are inherently "raw talent". There are much more factors that bat speed and swing path that equate to being a great hitter in the MLB.

✓ 3) Situational Pitching Strategy

How do pitch selection and pitch effectiveness (measured by opposing batter wOBA) vary by inning and game state (outs, inning, score difference, and base runners) on pitches that were hit? This is a two part question. First, on pitches that resulted in an out, can pitch selection be predicted based on outs, inning, score difference, and base runners? Second, with those different game states, which pitches are most effective based on estimated wOBA.

For part 1 of the question, it will likely be best to use decision trees. We have multiple features and they are categorical, despite being represented as numbers for some of them.

```
# set up df
df_situational = df[['pitch_name', 'estimated_woba_using_speedangle', 'inning', 'outs_when_up', 'pitcher', 'game_c

# Calculate score differential
df_situational['score_diff'] = np.abs(df_situational['home_score'] - df_situational['away_score'])

# Convert relevant columns to categorical types if needed
df_situational['inning'] = df_situational['inning'].astype('category')
df_situational['outs_when_up'] = df_situational['outs_when_up'].astype('category')
df_situational['score_diff'] = df_situational['score_diff'].astype('category')
df_situational['on_1b'] = df_situational['on_1b'].apply(lambda x: 0 if pd.isna(x) else 1) # Binary feature for ba
df_situational['on_2b'] = df_situational['on_2b'].apply(lambda x: 0 if pd.isna(x) else 1)
df_situational['on_3b'] = df_situational['on_3b'].apply(lambda x: 0 if pd.isna(x) else 1)

# Filter for only events that were outs for part 1 of question
df_situational_1 = df_situational[df_situational['events'].isin(['field_out', 'strikeout', 'sac_fly', 'grounded_ir
                        'force_out', 'sac_bunt', 'fielders_choice_out',
                        'fielders_choice', 'double_play', 'strikeout_double_play',
                        'triple_play', 'sac_fly_double_play'])].dropna()

# For part 2, remove na now that everything is converted. Only na should be in estimated wOBA, representing pitch
df_situational_2 = df_situational.dropna()
```


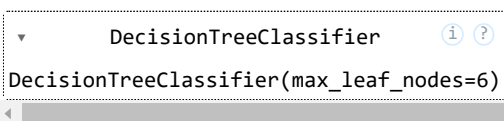
 [Show hidden output](#)

```
# Set up features and target variable
X = df_situational_1[['inning', 'outs_when_up', 'score_diff', 'on_1b', 'on_2b', 'on_3b']]
y = df_situational_1['pitch_name']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 562)

model = DecisionTreeClassifier(max_leaf_nodes=len(X.columns))

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

 0.2958077047588216

Based on the accuracy score, we were only able to predict roughly 29.58% of pitches correctly. With how low the accuracy score is, we can conclude that on pitches resulting in outs, we cannot predict what pitch will be thrown based on the game state. However, we can still look at part 2 of the question where we see which pitches in the different game states were most effective.

```
# Filter by desired pitches first
df_situational_2 = df_situational_2[df_situational_2['pitch_name'].isin(['4-Seam Fastball', 'Slider', 'Curveball', 'Sinker', 'Changeup', 'Split-Finger'])]


# For simplicity, we will split the base runners into RISP yes and no
df_situational_2['risp'] = df_situational_2.apply(lambda row: 1 if row['on_2b'] != 0 or row['on_3b'] != 0 else 0)

# Create combinations of the variables: 'outs_when_up' and 'risp'
# We'll group by the 6 combinations of outs and risp
grouped_woba = df_situational_2.groupby(['pitch_name', 'outs_when_up', 'risp'])['estimated_woba_using_speedangle']

# Create a new column to combine 'outs_when_up' and 'risp' for better visualization
grouped_woba['outs_risp_combo'] = grouped_woba['outs_when_up'].astype(str) + " out " + \
    grouped_woba['risp'].apply(lambda x: "RISP" if x == 1 else "No RISP")

pivot_woba = grouped_woba.pivot_table(
    index='outs_risp_combo', columns='pitch_name', values='estimated_woba_using_speedangle', aggfunc='mean')

pivot_woba
```

 <ipython-input-138-4b96db6b84c6>:10: FutureWarning: The default of observed=False is deprecated and will be

	pitch_name	4-Seam Fastball	Changeup	Curveball	Cutter	Sinker	Slider	Split-Finger	Sweeper
outs_risp_combo									
0 out	No RISP	0.345848	0.283979	0.265092	0.330191	0.353177	0.276398	0.238571	0.260229
	RISP	0.331653	0.320355	0.269556	0.328304	0.360183	0.275122	0.314498	0.284492
1 out	No RISP	0.333627	0.285286	0.293868	0.33835	0.350774	0.280052	0.249657	0.257087
	RISP	0.329534	0.295111	0.291385	0.346262	0.337996	0.293691	0.24942	0.267517
2 out	No RISP	0.331035	0.278024	0.273639	0.346621	0.343633	0.271317	0.252712	0.23962
	RISP	0.335031	0.307085	0.277564	0.32864	0.349607	0.28964	0.278904	0.266947

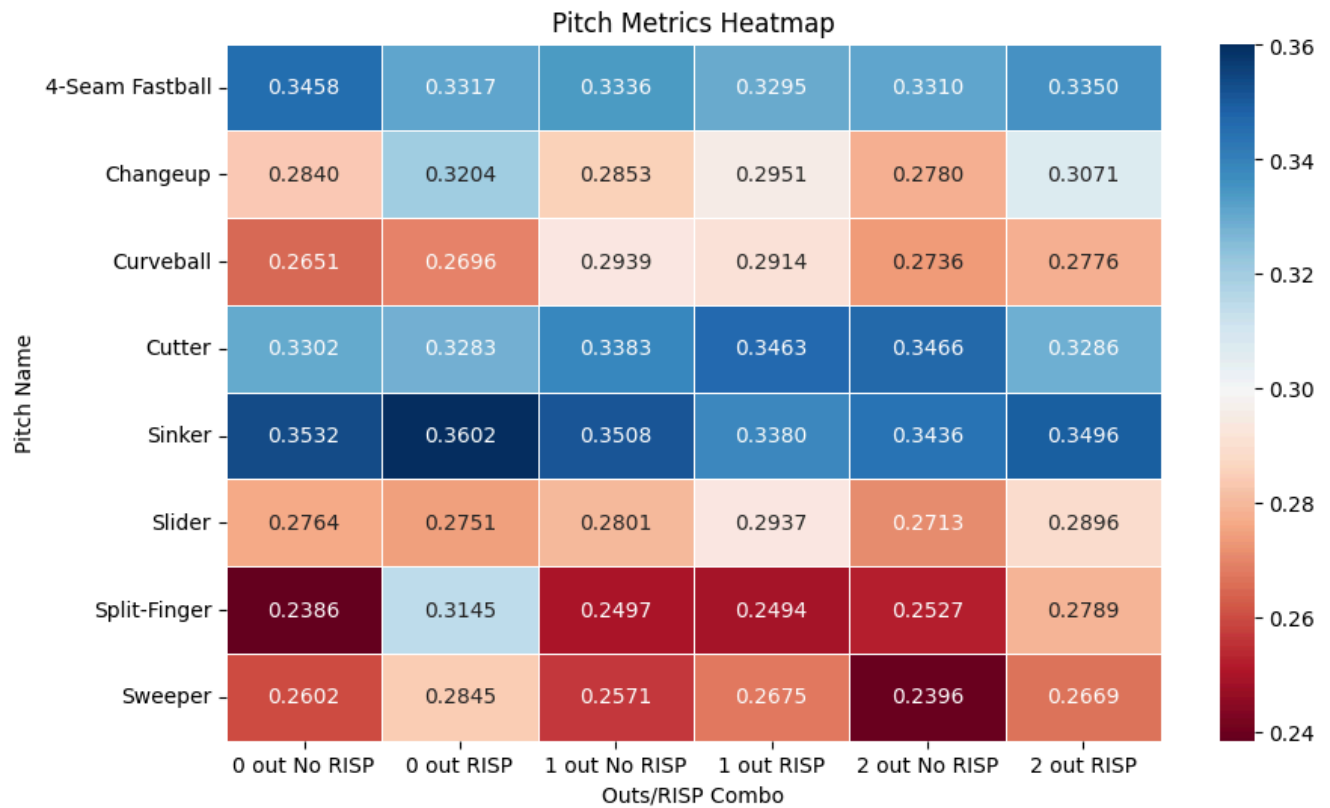
The best choice of visualization here I think will be a heatmap with one axis being the pitch names, and the other being the game state. The intersections will be the estimated wOBA. This will make it easy to see which pitch was best in which game state.

```
# Change pivot into df for ease of plotting
pivot_df = {
    'pitch_name': ['4-Seam Fastball', 'Changeup', 'Curveball', 'Cutter', 'Sinker', 'Slider', 'Split-Finger', 'Sw',
    '0 out No RISP': [0.345848, 0.283979, 0.265092, 0.330191, 0.353177, 0.276398, 0.238571, 0.260229],
    '0 out RISP': [0.331653, 0.320355, 0.269556, 0.328304, 0.360183, 0.275122, 0.314498, 0.284492],
    '1 out No RISP': [0.333627, 0.285286, 0.293868, 0.33835, 0.350774, 0.280052, 0.249657, 0.257087],
    '1 out RISP': [0.329534, 0.295111, 0.291385, 0.346262, 0.337996, 0.293691, 0.24942, 0.267517],
    '2 out No RISP': [0.331035, 0.278024, 0.273639, 0.346621, 0.343633, 0.271317, 0.252712, 0.23962],
    '2 out RISP': [0.335031, 0.307085, 0.277564, 0.32864, 0.349607, 0.28964, 0.278904, 0.266947]
}
pivot_df = pd.DataFrame(pivot_df)
pivot_df.set_index('pitch_name', inplace=True)

# Plot
plt.figure(figsize=(10, 6))
sns.heatmap(pivot_df, annot=True, cmap='RdBu', fmt=".4f", linewidths=0.5)

# Add labels and title
plt.title("Pitch Metrics Heatmap")
plt.ylabel("Pitch Name")
plt.xlabel("Outs/RISP Combo")

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



Based on the heatmap, we can see that there are trends across the different pitches, but not as much within each pitch. We see that the 4-seam fastball and sinker has a relatively high xwOBA regardless of game state, with the opposite holding true of split-finger and sweeper. There are some intersections that pop out, such as how xwOBA changes drastically for split-finger from 0 out No RISP to 0 out RISP. This information of course has to be taken with the fact that each pitch has a different usage rate and intent. Split-finger is primarily used as a strikeout pitch and will mostly be thrown with 2 strikes, inherently giving it better xwOBA.

Conclusion

Based on the accuracy of each of the models, it seems like the only research question that would be worth pursuing would be question 1. Ways this model could be enhanced and manipulated would be to use this as a player specific scouting model. Using data of just one batter, the model would be able to figure out what pitch type, speed, location, and movement would best induce a swing and miss from this batter. To further enhance, I could add a field that accounts for if the pitch is a righty or lefty. This could provide as a useful tool for researching schemes and gameplans for opposing batters.