1) While I have not worked directly with a sports team, I have significant experience in developing machine learning models to analyze player performance. I believe this expertise would translate well to a team environment. For example, I created a Jupyter notebook project that predicts pitchers' strikeout percentage (K%) and walk percentage (BB%) using advanced machine learning techniques. My XGBoost model demonstrated more predictive power for future K% and BB% than using those metrics alone, and I also developed a polynomial regression model with regularization that performed well in validation but lacked predictive power overall.

On the hitting side, I utilized Statcast data and batted ball statistics to predict hitters' weighted on-base average (wOBA). I included chase rate, zone contact percentage, pulled fly ball rate percentage, 90th percentile exit velocity, and an engineered feature called "damage," which is a spray angle extension of barrels. This feature considers the 80th percentile exit velocity at each spray angle for batted balls within the barrel classification limits of exit velocity and launch angle, providing a deeper understanding of how spray angle contributes to run production. This was included because it was a statistic found to be sticky year to year as measured by R2 score, and also was predictive of next season’s xwOBA.

While I haven't collaborated directly with coaches or players, my experience building these models positions me to contribute valuable insights to a team. By analyzing and applying key metrics, I can help inform decision-making and player development strategies, making data-driven approaches more actionable for a team's success.

<https://github.com/aflynn0213/K_BB_Evaluation>

2) Although I haven’t worked with Shiny specifically, I am a professional software engineer with 4+ years of experience, giving me a strong foundation in software development principles, including user interface design and application architecture. In addition to creating data visualizations and interactive tools through my Jupyter notebook projects—both for personal baseball analytics and my data science graduate studies—I have also developed graphical user interfaces (GUIs) for test environments at my job. This has helped me cultivate an engineering mindset when building intuitive, user-focused applications.

In my side projects, I have extensively used Python libraries such as Matplotlib, Seaborn, and Plotly to build data visualizations, including interactive dashboards to explore and present player performance metrics. For example, I’ve visualized relationships between Statcast metrics, such as zone contact percentage, exit velocities, and other performance indicators, to analyze and communicate player success.

While I haven’t had the opportunity to develop Shiny apps, I am confident that my experience in both software engineering and data visualization equips me to learn and utilize Shiny or any similar framework. My skills in creating dynamic, user-friendly interfaces and interactive visualizations would allow me to quickly adapt to building applications that enhance data accessibility and decision-making.

3) While I’ve already mentioned my baseball-related predictive modeling projects, my most extensive predictive algorithm was the movie recommendation engine I developed for my data science capstone project. This engine utilized collaborative filtering, where users could choose between two separate algorithms: K-Nearest Neighbors (KNN) or Singular Value Decomposition (SVD), depending on their preference.

KNN measures user similarity by comparing the movie ratings of different users. It identifies users (or "neighbors") who have rated movies similarly and recommends movies that those similar users enjoyed but the current user has not yet seen. The distance between users is typically measured using techniques like cosine similarity or Euclidean distance, allowing the algorithm to find the "nearest" users in terms of taste. By recommending movies based on the preferences of these neighbors, KNN delivers recommendations that align with the user’s past behavior.

SVD, on the other hand, takes a different approach. It decomposes the user-item rating matrix into three smaller matrices, capturing latent factors that influence movie preferences. This allows the model to find hidden relationships in the data, even when direct similarities between users are not apparent. For example, SVD might identify that certain groups of users tend to like specific genres, directors, or other features that are not explicitly labeled. This reduced matrix can then be used to predict missing ratings by multiplying the decomposed matrices back together, offering recommendations based on these hidden factors.

Both algorithms were implemented in Python, and the user could choose between them depending on their needs. I created a web interface using HTML and hosted it on a Flask server, allowing users to interact with the recommendation engine through a simple and intuitive platform. This project showcased my ability to build an end-to-end predictive system, combining machine learning techniques with practical web development to create a functional product.

<https://github.com/aflynn0213/MovieRecommenderForDummies>

4) To add substantial value to a baseball team, I would propose developing a model to better quantify and predict defensive value at first base, focusing specifically on range and the ability to save errant throws. Current defensive quantifying methods, such as UZR (Ultimate Zone Rating) and OAA (Outs Above Average), often fail to accurately represent the complexities of first base defense, leaving significant defensive contributions unaccounted for.

This project would begin by collecting Statcast data on fielding plays, specifically targeting first basemen's range and their ability to handle difficult throws from other infielders. The range component would involve mapping out zones on the field and calculating how frequently a first baseman successfully makes plays outside the expected catch zone. The ability to save errant throws would be modeled using both trajectory and velocity data to evaluate the difficulty of each throw and how frequently the first baseman prevents an error.

The methodology would include building a machine learning model, such as XGBoost, to predict defensive success based on these features. Key inputs would include throw velocity, angle, batted ball distance, first baseman positioning, and the first baseman's fielding metrics like reaction time and arm length.   
  
In addition to first base defense, catcher defensive metrics could also be enhanced significantly. By comparing a catcher's framing, blocking, and ability to throw out runners to the average catcher based on pitch type, velocity, horizontal and vertical movement, and location, we can identify areas for improvement. For instance, a catcher’s framing ability could be analyzed in relation to specific pitch types to determine how effective they are at turning borderline calls into strikes. Similarly, blocking metrics could account for pitch movement and location, providing a more nuanced understanding of a catcher's performance under varying conditions. This comprehensive approach would lead to more accurate assessments of both first basemen and catchers, ultimately contributing to improved team strategies and player development.

5) **Pitch Ranking (Quality of "Stuff"):**

1. **Slider (SL)**: This pitch ranks first due to its high spin rate (2675 RPM) and distinct movement profile. With 8.5 inches of horizontal break and slight vertical drop (-1.6 inches), the slider is likely to generate swings and misses. Though it has a low spin efficiency (38%), the high spin rate combined with horizontal movement can make it hard to hit, especially against same-handed batters. The combination of velocity and movement makes it an effective breaking pitch.
2. **Fastball (FB)**: The fastball ranks second, largely due to its moderate velocity (92.7 mph) and average spin rate (2145 RPM). Its spin efficiency (80%) and observed spin direction (1:25) create 13.4 inches of vertical rise and 8.5 inches of horizontal movement, which are solid but not elite. While it could be effective, particularly if well-located, the velocity is slightly below MLB average, which limits its overall potential.
3. **Changeup (CH)**: The changeup ranks last. Although it has a decent velocity differential from the fastball (8.2 mph slower), its spin rate is relatively low (1760 RPM), which could reduce deception. The changeup does have good horizontal movement (13.3 inches), but its vertical movement (9.1 inches) is not particularly impressive. Overall, this pitch is serviceable but could benefit from refinement.

### 

**Adjustments for Changeup:**

* More Drop: The pitcher should aim to increase the vertical drop by adjusting grip or release, possibly decreasing spin efficiency slightly to induce more vertical break. Bringing this number closer to 0 or slightly negative (around -2 to -4 inches) would likely improve the pitch's ability to get swings and misses, especially if paired with the fastball's ride.

**Consider Adding a Curveball or Cutter:**

* Curveball: A curveball would provide a breaking ball that can be effective against both left- and right-handed hitters. It offers vertical drop, adding contrast to the slider, which tends to have more horizontal movement. This would help the pitcher avoid platoon splits and give them an effective secondary option against batters of opposite handedness, since sliders tend to suffer from platoon splits.
* Cutter: A cutter could serve as a fastball variation that bridges the velocity and movement gap between the fastball and slider. With a cutter, the pitcher can better attack opposite-handed hitters, especially if the slider is less effective due to platoon splits. It would also give them a fastball against hitters of both handedness as the four seam profiles as a hybrid four-seamer and sinker due to its larger horizontal movement and non-elite ride, and sinkers tend also to suffer from platoon splits (most likely due to horizontal movement as seen also with sliders and sweepers. This gives the pitcher a fastball with less horizontal break and more late action, ideal for weak contact and a bridge between the fastball and slider.

6) I trained a model on all hitters with at least 250 PA from the seasons 2021-2024 using the six features presented in the table to predict next year's wRC+ in order to evaluate which skill set would lead to the highest production year over year. The predicted wRC+ values are:

* Player A: 107.768
* Player B: 103.951
* Player C: 114.749

Under the assumption that these players are of the same age, Player C’s baseline exit velocity is appealing, particularly if he’s young. Even with a detrimental, overly aggressive approach (50% swing rate and a 33% chase rate with poor out-of-zone contact skills), I believe this can be adjusted for a younger player. However, Player C's Z-Contact% of 75% is also troublesome, and the chase rate could be problematic as they age, as studies have shown that out-of-zone contact is one of the first skills to deteriorate with players past their prime. This suggests that players with high chase rates tend to age poorly. If Player C is past their prime, I would be less enthusiastic about acquiring them.

A comparison I found for Player C is O'Neil Cruz in 2024, but Cruz had an EV of 95.5 mph, 3rd among hitters, while our hypothetical player would rank 11th. On the other side of the aging curve but with a very comparable profile was Adolis Garcia., which speaks to the year over year volatility of a profile such as this. Still, that's a nice baseline to work with, and my simple polynomial regression model obviously heavily prioritized the nice exit velocity as it gave Player C the best prediction for next season wRC+ at 115. I also tend to agree with and understand the model since the power upside could potentially lead to the highest run production, and although 4 mph difference in EV doesn't seem drastic, if we once again look at the qualified hitters the next highest EV in our example of 89 mph would land at 80 on the leaderboards. At this spot we see an unweighted average wOBA among the three next hitters of .312 which is somewhat pedestrian.

Player A offers a hybrid of Alex Bregman and Luis Arraez due to the excellent contact skills (95% Z-Contact, 90% O-Contact) and moderate power (88.5 mph EV). This profile is intriguing regardless of age, as contact skills age well. Even with a 30% chase rate, Player A could afford to lose some out-of-zone contact and still remain productive with a historically great O-Contact% of 90%. The wRC+ of 107 as suggested by the model is likely a floor, with a higher ceiling if they can leverage more power through pulling fly balls, much like Bregman does.

Player B stands out for their strong plate discipline (25% O-Swing%) and respectable mid-level exit velocity (89.0 mph EV). Their contact skills are above average (85% Z-Contact and 60% O-Contact), and these traits are very repeatable and should age well. Despite the model predicting Player B’s wRC+ to be the lowest, I believe their skill set provides a solid floor with a 89 EV providing room for some upside. The model predicted this to be the lowest which surprises me, and therefore, rather than merely believing my model due to its obvious lack of context flaws, I'm going to go a different direction with my rankings.

Rankings:

1) Player A: The combination of elite contact skills, even with below-average to moderate power, gives Player A the potential for a high floor. By adjusting their spray angle to pull more fly balls, they could tap into additional power, which I believe is a feasible adjustment. Normally, I wouldn’t prioritize a contact-elitist profile, but this level of contact for a player with respectable exit velocity and chase rate would be historically exceptional, making Player A the top choice.

2) Player B: Player B’s strong plate discipline and repeatable skills make them a reliable option with a high floor. While their power ceiling is similar to Player A’s given the comparable exit velocity and launch angle, their superior on-base skills and above-average contact ability suggest that Player B could outperform their model-predicted wRC+ of 104, making them a close second to Player A.

3) Player C: While Player C has the highest power potential and predicted wRC+, their overly aggressive approach and poor contact skills raise concerns about how well they will age. If they are young and have time to refine their approach, their upside is significant, but the risk is higher, particularly in later years.

7) All the necessary code is provided in the Appendix section for this question. Individual plots for each unique pitcher is provided upon execution of the python script. Each plot contains pitcher pitch locations, pitch movement, and some stats for each individual pitch in the respective pitcher’s arsenal.

Appendix (Code):

For Question 6:

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

**from** sklearn**.**metrics **import** r2\_score**,** make\_scorer

**from** sklearn**.**preprocessing **import** PolynomialFeatures

**from** sklearn**.**linear\_model **import** Lasso**,** Ridge

**from** sklearn**.**pipeline **import** Pipeline

**from** sklearn**.**model\_selection **import** GridSearchCV

**def** normalize\_names**(**names**):**

normalized\_names **=** **[]**

**for** name **in** names**:**

# Replace special characters

name **=** name**.**replace**(**"Ã¡"**,** "a"**).**replace**(**"Ã©"**,** "e"**).**replace**(**"Ã­"**,** "i"**).**replace**(**"Ã³"**,** "o"**).**replace**(**"Ãº"**,** "u"**).**replace**(**"Ã±"**,** "n"**).**replace**(**"â€™"**,** "'"**).**replace**(**"â€˜"**,** "'"**)**

# Capitalize each word

name **=** ' '**.**join**(**word**.**capitalize**()** **for** word **in** name**.**split**())**

normalized\_names**.**append**(**name**)**

**return** normalized\_names

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Read the data from the specified sheet

df **=** pd**.**read\_csv**(**'player\_data.csv'**)** # Assuming 'LR' is the relevant sheet

# Normalize column names

df**.**columns **=** df**.**columns**.str.**lower**()**

# Normalize player names (if necessary)

df**[**'name'**]** **=** normalize\_names**(**df**[**'name'**])**

# Filter the DataFrame by seasons

df\_2021 **=** df**[**df**[**'season'**]** **==** 2021**]**

df\_2022 **=** df**[**df**[**'season'**]** **==** 2022**]**

df\_2023 **=** df**[**df**[**'season'**]** **==** 2023**]**

df\_2024 **=** df**[**df**[**'season'**]** **==** 2024**]**

# Set the index for each DataFrame

df\_2021**.**set\_index**([**'season'**,** 'name'**],** inplace**=True)**

df\_2022**.**set\_index**([**'season'**,** 'name'**],** inplace**=True)**

df\_2023**.**set\_index**([**'season'**,** 'name'**],** inplace**=True)**

df\_2024**.**set\_index**([**'season'**,** 'name'**],** inplace**=True)**

# Identify common players

players\_2021 **=** df\_2021**.**index**.**get\_level\_values**(**'name'**).**unique**()**

players\_2022 **=** df\_2022**.**index**.**get\_level\_values**(**'name'**).**unique**()**

players\_2023 **=** df\_2023**.**index**.**get\_level\_values**(**'name'**).**unique**()**

players\_2024 **=** df\_2024**.**index**.**get\_level\_values**(**'name'**).**unique**()**

common\_players\_2021\_2022 **=** **set(**players\_2021**).**intersection**(**players\_2022**)**

common\_players\_2022\_2023 **=** **set(**players\_2022**).**intersection**(**players\_2023**)**

common\_players\_2023\_2024 **=** **set(**players\_2023**).**intersection**(**players\_2024**)**

# Prepare the training data

# Features from 2021 to predict 2022 wRC+

x\_2021 **=** df\_2021**.**loc**[(**2021**,** **list(**common\_players\_2021\_2022**)),** **[**"o-swing% (sc)"**,** "swing% (sc)"**,** "o-contact% (sc)"**,** "z-contact% (sc)"**,** "ev"**,** "la"**]]**

y\_2022 **=** df\_2022**.**loc**[(**2022**,** **list(**common\_players\_2021\_2022**)),** "wrc+"**]**

# Features from 2022 to predict 2023 wRC+

x\_2022 **=** df\_2022**.**loc**[(**2022**,** **list(**common\_players\_2022\_2023**)),** **[**"o-swing% (sc)"**,** "swing% (sc)"**,** "o-contact% (sc)"**,** "z-contact% (sc)"**,** "ev"**,** "la"**]]**

y\_2023 **=** df\_2023**.**loc**[(**2023**,** **list(**common\_players\_2022\_2023**)),** "wrc+"**]**

x\_2023 **=** df\_2023**.**loc**[(**2023**,** **list(**common\_players\_2023\_2024**)),** **[**"o-swing% (sc)"**,** "swing% (sc)"**,** "o-contact% (sc)"**,** "z-contact% (sc)"**,** "ev"**,** "la"**]]**

y\_2024 **=** df\_2024**.**loc**[(**2024**,** **list(**common\_players\_2023\_2024**)),** "wrc+"**]**

# Sort the indices

x\_2021 **=** x\_2021**.**sort\_index**(**level**=**'name'**)**

y\_2022 **=** y\_2022**.**sort\_index**(**level**=**'name'**)**

x\_2022 **=** x\_2022**.**sort\_index**(**level**=**'name'**)**

y\_2023 **=** y\_2023**.**sort\_index**(**level**=**'name'**)**

x\_2023 **=** x\_2023**.**sort\_index**(**level**=**'name'**)** # New line for sorting 2023

y\_2024 **=** y\_2024**.**sort\_index**(**level**=**'name'**)**

# Combine training features and targets

x\_train **=** pd**.**concat**([**x\_2021**,** x\_2022**,** x\_2023**])** # Combine features for 2021, 2022, and 2023

y\_train **=** pd**.**concat**([**y\_2022**,** y\_2023**,** y\_2024**])**

# Display the shapes of the training sets

**print(**f"Training features shape: {x\_train**.**shape}"**)**

**print(**f"Training target shape: {y\_train**.**shape}"**)**

# Set up the pipeline

pipeline **=** Pipeline**([**

**(**'poly'**,** PolynomialFeatures**(**include\_bias**=True)),**

**(**'regressor'**,** Lasso**())**

**])**

# Define parameter grid for GridSearchCV

param\_grid **=** **{**

'regressor'**:** **[**Lasso**(),** Ridge**()],**

'poly\_\_degree'**:** **[**1**,** 2**,** 3**,** 4**],**

'poly\_\_interaction\_only'**:** **[False],**

'poly\_\_include\_bias'**:** **[True,** **False],**

'regressor\_\_alpha'**:** **[**0.001**,** 0.01**,** 0.1**,** 0.25**,** 1.0**,** 2.5**,** 5**,** 10.0**,** 20**]**

**}**

# Scorer

scorer **=** make\_scorer**(**r2\_score**)**

# Perform GridSearchCV

model **=** GridSearchCV**(**pipeline**,** param\_grid**,** scoring**=**scorer**,** n\_jobs**=-**1**,** cv**=**4**)**

model**.**fit**(**x\_train**,** y\_train**)**

# Extract the best estimator and parameters

best\_estimator **=** model**.**best\_estimator\_

best\_regressor **=** best\_estimator**.**named\_steps**[**'regressor'**]**

**print(**f"Best regressor: {best\_regressor**.**\_\_class\_\_**.**\_\_name\_\_}"**)**

**print(**f"Best training parameters: {best\_regressor**.**get\_params**()**}"**)**

**print(**f"Best poly training parameters: {best\_estimator**.**named\_steps**[**'poly'**].**get\_params**()**}"**)**

**print(**f"Best score: {model**.**best\_score\_}"**)**

# Here, you can proceed to test on future data or other analysis as needed

y\_pred **=** best\_estimator**.**predict**(**x\_train**)**

# Create a DataFrame with player names and predicted wRC+

predictions\_df **=** pd**.**DataFrame**({**

'Predicted wRC+'**:** y\_pred

**},** index**=**x\_train**.**index**)**

# Display the predictions DataFrame

**print(**predictions\_df**)**

# Original data

data **=** **{**

'Season'**:** **[**'2024'**,** '2024'**,** '2024'**],**

'Name'**:** **[**'Player A'**,** 'Player B'**,** 'Player C'**],**

'O-Swing%'**:** **[**0.30**,** 0.25**,** 0.33**],** # Decimals instead of percentage strings

'Swing%'**:** **[**0.40**,** 0.45**,** 0.50**],** # Decimals instead of percentage strings

'O-Contact%'**:** **[**0.90**,** 0.60**,** 0.55**],** # Decimals instead of percentage strings

'Z-Contact%'**:** **[**0.95**,** 0.85**,** 0.75**],** # Decimals instead of percentage strings

'EV'**:** **[**88.5**,** 89.0**,** 93.0**],**

'LA'**:** **[**12.5**,** 13.0**,** 12.0**]**

**}**

# Create a DataFrame

df **=** pd**.**DataFrame**(**data**)** # Remove the index argument

df**.**columns **=** df**.**columns**.str.**lower**()**

# Make predictions for the example players

wrc\_predictions **=** best\_estimator**.**predict**(**df**[[**"o-swing%"**,** "swing%"**,** "o-contact%"**,** "z-contact%"**,** "ev"**,** "la"**]])**

# Display the predicted wRC+

prediction\_results **=** pd**.**DataFrame**({**'Predicted wRC+'**:** wrc\_predictions**},** index**=**df**.**name**)**

**print(**prediction\_results**)**

For Question 7:

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

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

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

**from** matplotlib**.**gridspec **import** GridSpec

**from** matplotlib**.**patches **import** Rectangle

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

**def** render\_pitcher\_analysis**(**pitcher\_data**,** name**):**

# Change the style to a more modern look

plt**.**style**.**use**(**'seaborn-darkgrid'**)** # Changed style

plt**.**rcParams**[**'axes.grid'**]** **=** **True**

plt**.**rcParams**[**'grid.linestyle'**]** **=** '--' # Changed to dashed lines

plt**.**rcParams**[**'grid.color'**]** **=** '#B0B0B0' # Light gray color

# Create a figure with different size for better layout

fig **=** plt**.**figure**(**figsize**=(**28**,** 24**))** # Changed size for a more rectangular layout

gs **=** GridSpec**(**4**,** 3**,** figure**=**fig**,** height\_ratios**=[**0.15**,** 1**,** 1.5**,** 0.75**])** # Adjusted ratios for better spacing

fig**.**patch**.**set\_linewidth**(**3**)** # Thicker border

fig**.**patch**.**set\_edgecolor**(**'#404040'**)** # Darker border color

pitch\_color\_mapping **=** **{**

'CH'**:** '#ff7f0e'**,** 'SL'**:** '#ff9896'**,** 'CU'**:** '#d62728'**,** 'FS'**:** '#bcbd22'**,**

'FF'**:** '#1f77b4'**,** 'SC'**:** '#2ca02c'**,** 'SI'**:** '#17becf'**,** 'SV'**:** '#ffbb78'**,**

'FO'**:** '#7f7f7f'**,** 'KC'**:** '#8c564b'**,** 'ST'**:** '#c5b0d5'**,** 'FC'**:** '#9467bd'**,**

'CS'**:** '#e377c2'

**}**

ax\_overview **=** fig**.**add\_subplot**(**gs**[**0**,** **:])**

display\_count\_statistics**(**pitcher\_data**,** ax\_overview**,** fontsize**=**25**)**

ax\_left **=** fig**.**add\_subplot**(**gs**[**1**,** 0**])**

plot\_pitch\_locations**(**pitcher\_data**[**pitcher\_data**[**'BatterSide'**]** **==** 'L'**],** ax\_left**,** 'LHH'**,** pitch\_color\_mapping**)**

ax\_right **=** fig**.**add\_subplot**(**gs**[**1**,** 1**])**

plot\_pitch\_locations**(**pitcher\_data**[**pitcher\_data**[**'BatterSide'**]** **==** 'R'**],** ax\_right**,** 'RHH'**,** pitch\_color\_mapping**)**

ax\_break **=** fig**.**add\_subplot**(**gs**[**1**,** 2**])**

plot\_pitch\_break\_analysis**(**pitcher\_data**,** ax\_break**,** pitch\_color\_mapping**)**

ax\_pitches **=** fig**.**add\_subplot**(**gs**[**2**,** **:])**

display\_pitch\_statistics**(**pitcher\_data**,** ax\_pitches**,** fontsize**=**25**)**

pitcher\_hand **=** pitcher\_data**[**'PitcherHand'**].**iloc**[**0**]**

handedness\_label **=** "LHP" **if** pitcher\_hand **==** "L" **else** "RHP"

fig**.**suptitle**(**f"{name} ({handedness\_label})"**,** fontsize**=**35**,** y**=**0.98**)**

plt**.**tight\_layout**()**

plt**.**subplots\_adjust**(**top**=**0.95**,** bottom**=**0.05**,** hspace**=**0.4**)**

plt**.**show**()**

def plot\_pitch\_locations(data, ax, title, pitch\_color\_map):

avg\_locations = data.groupby('PitchType').agg(

avg\_x=('TrajectoryLocationX', 'mean'),

avg\_z=('TrajectoryLocationZ', 'mean')

).reset\_index()

for \_, row in avg\_locations.iterrows():

ax.scatter(row['avg\_x'], row['avg\_z'],

label=row['PitchType'],

alpha=0.7,

s=400,

color=pitch\_color\_map.get(row['PitchType'], 'gray'))

zone\_bottom = data['StrikeZoneBottom'].mean()

zone\_top = data['StrikeZoneTop'].mean()

zone\_width = 1.4166

zone\_height = zone\_top - zone\_bottom

zone\_center = 0

strike\_zone = Rectangle((zone\_center - zone\_width / 2, zone\_bottom), zone\_width, zone\_height,

fill=False, color='k', linewidth=2)

ax.add\_patch(strike\_zone)

plate\_width = zone\_width

plate = plt.Polygon([(-plate\_width / 2, 0), (0, 0), (plate\_width / 2, 0)], color='k')

ax.add\_patch(plate)

padding\_x = 0.5

padding\_y = 0.5

ax.set\_xlim((zone\_center - zone\_width / 2) - padding\_x, (zone\_center + zone\_width / 2) + padding\_x)

ax.set\_ylim(zone\_bottom - padding\_y, zone\_top + padding\_y)

ax.set\_xlabel('', fontsize=18)

ax.set\_ylabel('', fontsize=18)

ax.set\_title(f'{title}', fontsize=20)

handles, labels = ax.get\_legend\_handles\_labels()

ax.legend(handles, labels, title="Pitch Type", loc="upper right", fontsize=8, title\_fontsize='12')

ax.invert\_xaxis()

ax.set\_xticks([])

ax.set\_yticks([])

ax.set\_facecolor('white')

ax.grid(False)

for spine in ax.spines.values():

spine.set\_visible(False)

ax.set\_aspect(1.02)

def plot\_pitch\_break\_analysis(data, ax, pitch\_color\_map):

data['TrajectoryHorizontalBreak'] \*= 12

data['TrajectoryVerticalBreakInduced'] \*= 12

for pitch\_type in data['PitchType'].unique():

pitch\_data = data[data['PitchType'] == pitch\_type]

ax.scatter(pitch\_data['TrajectoryHorizontalBreak'],

pitch\_data['TrajectoryVerticalBreakInduced'],

label=pitch\_type,

alpha=0.7,

s=70,

color=pitch\_color\_map.get(pitch\_type, 'gray'))

ax.set\_xlabel('Horizontal Break (in)', fontsize=26)

ax.set\_ylabel('Vertical Break (in)', fontsize=26)

ax.set\_title('Pitch Break', fontsize=28)

handles, labels = ax.get\_legend\_handles\_labels()

ax.legend(handles, labels, title="Pitch Type", loc="upper right", fontsize=14, title\_fontsize='16')

ax.axhline(y=0, color='k', linestyle='--', linewidth=0.5)

ax.axvline(x=0, color='k', linestyle='--', linewidth=0.5)

max\_horz = data['TrajectoryHorizontalBreak'].max() \* 1.1

min\_horz = data['TrajectoryHorizontalBreak'].min() \* 1.1

max\_vert = data['TrajectoryVerticalBreakInduced'].max() \* 1.1

min\_vert = data['TrajectoryVerticalBreakInduced'].min() \* 1.1

ax.set\_xlim(min\_horz, max\_horz)

ax.set\_ylim(min\_vert, max\_vert)

ax.tick\_params(axis='both', which='major', labelsize=14)

def display\_count\_statistics(pitcher\_data, ax, fontsize=25):

batters\_faced = pitcher\_data['AtBatNumber'].nunique()

strikeouts = (pitcher\_data['PitchCall'] == 'strikeout').sum()

walks = (pitcher\_data['PitchCall'] == 'walk').sum()

singles = (pitcher\_data['PitchCall'] == 'single').sum()

doubles = (pitcher\_data['PitchCall'] == 'double').sum()

triples = (pitcher\_data['PitchCall'] == 'triple').sum()

home\_runs = (pitcher\_data['PitchCall'] == 'home\_run').sum()

hits = singles + doubles + triples + home\_runs

at\_bats = batters\_faced - walks - (pitcher\_data['PitchCall'] == 'hit\_by\_pitch').sum()

total\_bases = singles + (2 \* doubles) + (3 \* triples) + (4 \* home\_runs)

valid\_pitches = pitcher\_data['PitchType'].notna().sum()

outs = (

strikeouts +

(pitcher\_data['PitchCall'] == 'field\_out').sum() +

(pitcher\_data['PitchCall'] == 'sac\_bunt').sum() +

(pitcher\_data['PitchCall'] == 'force\_out').sum() +

2 \* (pitcher\_data['PitchCall'] == 'grounded\_into\_double\_play').sum()

)

innings\_pitched = f"{outs // 3}"

if outs % 3 == 1:

innings\_pitched += " 1/3"

elif outs % 3 == 2:

innings\_pitched += " 2/3"

WHIP = float(walks+hits)/float(innings\_pitched)

count\_stats = {

'Pitches': valid\_pitches,

'PA': batters\_faced,

'Ks': strikeouts,

'BBs': walks,

'HRs': home\_runs,

'Hits': hits,

'WHIP': WHIP,

'Opp SLG': f"{(total\_bases / at\_bats):.3f}" if at\_bats > 0 else "0.000"

}

count\_stats['IP'] = innings\_pitched

df\_counts = pd.DataFrame([count\_stats])

table = ax.table(cellText=df\_counts.values, colLabels=df\_counts.columns,

cellLoc='center', loc='center')

table.auto\_set\_font\_size(True)

table.scale(1, 4)

ax.axis('off')

def display\_pitch\_statistics(pitcher\_data, ax, fontsize=25):

pitcher\_data['true\_spin'] = np.sqrt(

pitcher\_data['SpinVectorX']\*\*2 +

pitcher\_data['SpinVectorZ']\*\*2

)

pitcher\_data['spin\_efficiency'] = (pitcher\_data['true\_spin'] / pitcher\_data['ReleaseSpinRate']) \* 100

pitcher\_data['spin\_efficiency'] = np.clip(pitcher\_data['spin\_efficiency'], 0, 100)

at\_bat\_outcomes = ['single', 'double', 'triple', 'home\_run', 'field\_out', 'strikeout', 'walk']

pitch\_counts = pitcher\_data['PitchType'].value\_counts()

pitch\_stats = pitcher\_data.groupby('PitchType').agg(

count=('PitchType', 'size'),

avg\_velocity=('ReleaseSpeed', 'mean'),

avg\_spin\_rate=('ReleaseSpinRate', 'mean'),

avg\_spin\_efficiency=('spin\_efficiency', 'mean'),

avg\_pitching\_hand=('PitcherHand', 'first'),

avg\_vertical\_break=('TrajectoryVerticalBreakInduced', 'mean'),

avg\_horizontal\_break=('TrajectoryHorizontalBreak', 'mean'),

whiff\_percentage=('PitchCall', lambda x: sum(x == 'swinging\_strike') / len(x))

).reset\_index()

pitch\_stats['spin\_efficiency'] = pitch\_stats['avg\_spin\_efficiency'].round(1)

pitch\_stats['avg\_velocity'] = pitch\_stats['avg\_velocity'].round(1)

pitch\_stats['avg\_spin\_rate'] = pitch\_stats['avg\_spin\_rate'].round(1)

pitch\_stats['avg\_batter\_hand'] = pitcher\_data['BatterSide'].value\_counts()

pitch\_stats['avg\_batter\_hand'] = pitch\_stats['avg\_batter\_hand'].get('R', 0)

pitch\_stats['whiff\_rate'] = pitch\_stats['whiff\_percentage'].apply(lambda x: f"{x:.1%}")

pitch\_stats = pitch\_stats[['PitchType', 'count', 'avg\_velocity', 'avg\_spin\_rate', 'spin\_efficiency', 'avg\_vertical\_break', 'avg\_horizontal\_break', 'whiff\_rate']]

columns = pitch\_stats.columns

table = ax.table(cellText=pitch\_stats.values, colLabels=pitch\_stats.columns,

cellLoc='center', bbox=[0, 0, 1, 1])

table.auto\_set\_font\_size(False)

table.set\_fontsize(fontsize)

table.scale(1, 1.8)

ax.axis('off')

def display\_batter\_hand\_analysis(pitcher\_data, ax, title):

hand\_counts = pitcher\_data['BatterSide'].value\_counts()

ax.pie(hand\_counts, labels=hand\_counts.index, autopct='%1.1f%%', startangle=90)

ax.axis('equal')

ax.set\_title(title)# Load the data

if \_\_name\_\_ == '\_\_main\_\_':

pitch\_data = pd.read\_csv("AnalyticsQuestionnairePitchData.csv")

for uniq in pitch\_data['PitcherId'].unique():

render\_pitcher\_analysis(pitch\_data[pitch\_data['PitcherId'] == uniq], f"Pitcher {uniq}")