An Analysis of MLB Pitch Selection

Brendan Karadenes

Table of contents

Introduction

The purpose of this project was to examine how Major League Baseball (MLB) pitchers react to different in-game scenarios. To find an answer to this question we looked at each pitcher's pitch arsenals and tendencies. To maintain consistency, we used only qualifying pitchers (1 inning pitched per team game) from the 2023 season. All of the data was retrieved from MLB's Statcast database where we were able to see each pitcher's pitch from the season.In addition, the result of each pitch and other advanced metrics were available to view. To begin the investigation we separated each pitcher into 5 different clusters based on their pitching arsenal. We looked at pitch selection for each group and were able to visualize how each type of pitcher reacted to different situations, like different outs and counts. We compared pitchers both within and between different clusters, making a small shiny app to visualize this. Then, we built various models for each pitcher to see if certain statistics, like Earned Run Average (ERA) could predict how much the pitcher deviated from their primary pitch.

Step 1: Pitcher Clustering

To begin the project, we gathered each qualified pitcher and their 2023 season statistics into a single dataset. The list of pitchers can be viewed below:

Loading in Pitchers

R Version

```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
```

```
library(gt)
library(tidyverse)
players <- data.frame(
  Player = c("Webb, Logan",
  "Gallen, Zac",
  "Cole, Gerrit",
  "Mikolas, Miles",
  "Bassitt, Chris",
  "Valdez, Framber",
  "Castillo, Luis",
  "Keller, Mitch",
  "López, Pablo",
  "Burnes, Corbin",
  "Nola, Aaron",
  "Gibson, Kyle",
  "Wheeler, Zack",
  "Gilbert, Logan",
  "Kirby, George",
  "Berríos, José",
  "Montgomery, Jordan",
  "Strider, Spencer",
  "Gausman, Kevin",
  "Alcantara, Sandy",
  "Giolito, Lucas",
  "Gray, Sonny",
  "Lynn, Lance",
  "Corbin, Patrick",
  "Snell, Blake",
  "Luzardo, Jesús",
  "Eflin, Zach",
  "Kelly, Merrill",
  "Lyles, Jordan",
  "Oviedo, Johan",
  "Cease, Dylan",
  "Elder, Bryce",
  "Steele, Justin",
```

```
  "Dunning, Dane",
  "Kremer, Dean",
  "Walker, Taijuan",
  "Sears, JP",
  "Bradish, Kyle",
  "Kikuchi, Yusei",
  "Peralta, Freddy",
  "Morton, Charlie",
  "Verlander, Justin",
  "Javier, Cristian")
)
players_table <- players %>%
  gt() %>%
  tab_header(title = "Qualified Pitchers")
players_table
```

The code below reads in player statistics from qualified pitchers in 2023 and places them in a data frame.

```
qual_pitchers <- read.csv("qual_pitchers.csv")
```

```
qual_pitchers %>%
  summarise(unique_count = n_distinct(pitcher))
```

```
  unique_count
1          446
```

```
filtered_pitchers <- qual_pitchers %>%
filter(player_name %in% c(
  "Webb, Logan",
  "Gallen, Zac",
  "Cole, Gerrit",
  "Mikolas, Miles",
  "Bassitt, Chris",
  "Valdez, Framber",
  "Castillo, Luis",
  "Keller, Mitch",
  "López, Pablo",
  "Burnes, Corbin",
  "Nola, Aaron",
```

Qualified Pitchers

Player
Webb, Logan
Gallen, Zac
Cole, Gerrit
Mikolas, Miles
Bassitt, Chris
Valdez, Framber
Castillo, Luis
Keller, Mitch
López, Pablo
Burnes, Corbin
Nola, Aaron
Gibson, Kyle
Wheeler, Zack
Gilbert, Logan
Kirby, George
Berríos, José
Montgomery, Jordan
Strider, Spencer
Gausman, Kevin
Alcantara, Sandy
Giolito, Lucas
Gray, Sonny
Lynn, Lance
Corbin, Patrick
Snell, Blake
Luzardo, Jesús
Eflin, Zach
Kelly, Merrill
Lyles, Jordan
Oviedo, Johan
Cease, Dylan
Elder, Bryce
Steele, Justin
Dunning, Dane
Kremer, Dean
Walker, Taijuan
Sears, JP
Bradish, Kyle
Kikuchi, Yusei
Peralta, Freddy
Morton, Charlie
Verlander, Justin
Javier, Cristian

```
  "Gibson, Kyle",
  "Wheeler, Zack",
  "Gilbert, Logan",
  "Kirby, George",
  "Berríos, José",
  "Montgomery, Jordan",
  "Strider, Spencer",
  "Gausman, Kevin",
  "Alcantara, Sandy",
  "Giolito, Lucas",
  "Gray, Sonny",
  "Lynn, Lance",
  "Corbin, Patrick",
  "Snell, Blake",
  "Luzardo, Jesús",
  "Eflin, Zach",
  "Kelly, Merrill",
  "Lyles, Jordan",
  "Oviedo, Johan",
  "Cease, Dylan",
  "Elder, Bryce",
  "Steele, Justin",
  "Dunning, Dane",
  "Kremer, Dean",
  "Walker, Taijuan",
  "Sears, JP",
  "Bradish, Kyle",
  "Kikuchi, Yusei",
  "Peralta, Freddy",
  "Morton, Charlie",
  "Verlander, Justin",
  "Javier, Cristian"
))
```

Python Version

```
install.packages("reticulate")
```

```
The following package(s) will be installed:
- reticulate [1.42.0]
These packages will be installed into "C:/Users/bkara/OneDrive - St. Lawrence University/SYE/
```

```
# Installing packages --------------------------------------------------------
- Installing reticulate ...                        OK [linked from cache]
Successfully installed 1 package in 21 milliseconds.
```

```r
library(reticulate)
```

```python
import pandas as pd
from plotnine import *
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import fcluster
players = pd.DataFrame({
    "Player": [
        "Webb, Logan", "Gallen, Zac", "Cole, Gerrit", "Mikolas, Miles",
        "Bassitt, Chris", "Valdez, Framber", "Castillo, Luis", "Keller, Mitch",
        "López, Pablo", "Burnes, Corbin", "Nola, Aaron", "Gibson, Kyle",
        "Wheeler, Zack", "Gilbert, Logan", "Kirby, George", "Berríos, José",
        "Montgomery, Jordan", "Strider, Spencer", "Gausman, Kevin", "Alcantara, Sandy",
        "Giolito, Lucas", "Gray, Sonny", "Lynn, Lance", "Corbin, Patrick",
        "Snell, Blake", "Luzardo, Jesús", "Eflin, Zach", "Kelly, Merrill",
        "Lyles, Jordan", "Oviedo, Johan", "Cease, Dylan", "Elder, Bryce",
        "Steele, Justin", "Dunning, Dane", "Kremer, Dean", "Walker, Taijuan",
        "Sears, JP", "Bradish, Kyle", "Kikuchi, Yusei", "Peralta, Freddy",
        "Morton, Charlie", "Verlander, Justin", "Javier, Cristian"
    ]
})
```

```python
qual_pitchers = pd.read_csv("qual_pitchers.csv")
```

```python
unique_pitchers_count = qual_pitchers['pitcher'].nunique()
```

```python
filtered_pitchers = qual_pitchers[qual_pitchers['player_name'].isin(players['Player'])]
```

Pitch Proportions

To prepare for the clustering we gathered the proportion of each pitch type for each pitcher. This way, we can compare pitch selection between them. We also need to create the `count` variable to see in what count each pitch was thrown.

R Version

```r
filtered_pitchers <- filtered_pitchers %>%
  unite(count, balls, strikes, sep = "-")
```

```r
grouped_pitches <- filtered_pitchers %>%
  group_by(player_name, pitch_type, count) %>%
  summarise(pitch_count = n(), .groups = 'drop') %>%
  group_by(player_name, count) %>%
  mutate(total_pitches_in_count = sum(pitch_count),
         pitch_proportion = pitch_count / total_pitches_in_count) %>%
  ungroup()
```

Python Version

```python
filtered_pitchers['count'] = filtered_pitchers['balls'].astype(str) + "-" + filtered_pitchers
```

```python
grouped_pitches = (
    filtered_pitchers.groupby(['player_name', 'pitch_type', 'count'])
    .size()
    .reset_index(name='pitch_count')
)

grouped_pitches['total_pitches_in_count'] = (
    grouped_pitches.groupby(['player_name', 'count'])['pitch_count'].transform('sum')
)

grouped_pitches['pitch_proportion'] = grouped_pitches['pitch_count'] / grouped_pitches['total
```

Situational Pitch Usage

In addition, within each cluster, we looked at how pitchers used each pitch within each count. This was an important step because we were able to see any changes (or lack thereof) in different situations. The code above groups the data by pitcher, pitch, and count and calculates the number of pitches thrown for each combination of the three variables. Using the `pitch_proportion` variable, we are able to see the proportion of that pitch relative to all pitches thrown in that count.

R Version

```
grouped_pitches_by_outs <- filtered_pitchers %>%
  group_by(player_name, pitch_type, outs_when_up) %>%
  summarise(pitch_count = n(), .groups = 'drop') %>%
  group_by(player_name, outs_when_up) %>%
  mutate(total_pitches_in_outs = sum(pitch_count),
         pitch_proportion = pitch_count / total_pitches_in_outs) %>%
  ungroup()

grouped_pitches_by_outs
```

```
# A tibble: 505 x 6
   player_name    pitch_type outs_when_up pitch_count total_pitches_in_outs
   <chr>          <chr>             <int>       <int>                 <int>
 1 Bassitt, Chris CH                    0           2                    32
 2 Bassitt, Chris CH                    1           7                    39
 3 Bassitt, Chris CH                    2           1                    35
 4 Bassitt, Chris CU                    0           3                    32
 5 Bassitt, Chris CU                    1           2                    39
 6 Bassitt, Chris CU                    2           8                    35
 7 Bassitt, Chris FC                    0           4                    32
 8 Bassitt, Chris FC                    1           6                    39
 9 Bassitt, Chris FC                    2           7                    35
10 Bassitt, Chris FF                    0           2                    32
# i 495 more rows
# i 1 more variable: pitch_proportion <dbl>
```

The code above is similar to the chunk working with different counts. Instead, we are grouping player name and pitch type with the number of outs and calculating the proportion of pitches thrown in each outs situation (0-2).

The code below makes a column for each pitch type and their proportion thrown with a given number of outs. The second chunk makes it so the columns now have wieghted pitch proportions for each pitch type, which will reduce the impact of small sample sizes.

```
pitchers_grouped <- grouped_pitches_by_outs %>%
  pivot_wider(names_from = pitch_type, values_from = pitch_proportion, values_fill = 0)
```

```
aggregated_pitchers_by_outs <- pitchers_grouped %>%
  group_by(player_name, outs_when_up) %>%
  summarise(across(starts_with("CH"):starts_with("KC"),
```

```
                    ~ sum(. * total_pitches_in_outs) / sum(total_pitches_in_outs),
                    .names = "weighted_{col}")) %>%
  ungroup()

aggregated_pitchers_by_outs
```

```
# A tibble: 114 x 12
   player_name    outs_when_up weighted_CH weighted_CU weighted_FC weighted_FF
   <chr>                 <int>       <dbl>       <dbl>       <dbl>       <dbl>
 1 Bassitt, Chris            0     0.0125      0.0188       0.025      0.0125
 2 Bassitt, Chris            1     0.0299      0.00855      0.0256     0
 3 Bassitt, Chris            2     0.00571     0.0457       0.04       0.00571
 4 Berríos, José             0     0.0667      0            0          0.0556
 5 Berríos, José             1     0.0517      0            0          0.0345
 6 Berríos, José             2     0.0583      0            0          0.025
 7 Bradish, Kyle             0     0.00526     0.0211       0          0.0316
 8 Bradish, Kyle             1     0.0174      0.0522       0          0.0261
 9 Bradish, Kyle             2     0.0370      0.0667       0          0.0667
10 Burnes, Corbin            0     0.00893     0.0357       0.179      0
# i 104 more rows
# i 6 more variables: weighted_FS <dbl>, weighted_SI <dbl>, weighted_SL <dbl>,
#   weighted_ST <dbl>, weighted_SV <dbl>, weighted_KC <dbl>
```

Python Version

```
# Count pitches by player, pitch type, and outs
grouped_pitches_by_outs = (
    filtered_pitchers
    .groupby(['player_name', 'pitch_type', 'outs_when_up'])
    .size()
    .reset_index(name='pitch_count')
)

# Add total pitches for each player and outs situation
grouped_pitches_by_outs['total_pitches_in_outs'] = (
    grouped_pitches_by_outs
    .groupby(['player_name', 'outs_when_up'])['pitch_count']
    .transform('sum')
)

# Calculate proportion of each pitch type
```

```
grouped_pitches_by_outs['pitch_proportion'] = (
    grouped_pitches_by_outs['pitch_count'] /
    grouped_pitches_by_outs['total_pitches_in_outs']
)

# Pivot so each pitch type is a column (like pivot_wider in R)
pitchers_grouped = grouped_pitches_by_outs.pivot_table(
    index=['player_name', 'outs_when_up'],
    columns='pitch_type',
    values='pitch_proportion',
    fill_value=0
).reset_index()

# Merge total_pitches_in_outs back in
pitch_counts = (
    grouped_pitches_by_outs[['player_name', 'outs_when_up', 'total_pitches_in_outs']]
    .drop_duplicates()
)
pitchers_grouped = pd.merge(pitchers_grouped, pitch_counts, on=['player_name', 'outs_when_up

# Apply weighted average function
pitch_cols = ['CH', 'CU', 'FC', 'FF', 'FS', 'FT', 'KC', 'SI', 'SL']

def weighted_avg(group):
    weights = group['total_pitches_in_outs']
    return pd.Series({
        f'weighted_{col}': (group[col] * weights).sum() / weights.sum()
        for col in pitch_cols if col in group.columns
    })

aggregated_pitchers_by_outs = (
    pitchers_grouped
    .groupby(['player_name', 'outs_when_up'])
    .apply(weighted_avg)
    .reset_index()
)
```

Hierarchical Clustering

To visualize these results we looked at various different visuals including hierarchical clustering trees and heat maps. This proved to be difficult because with the amount of different pitchers

the output was messy and difficult to see the results. One of the solutions to this problem was cutting the tree which cut down the number of clusters and made larger groups of similar pitchers. Viewing pitch usage by cluster provided a baseline on how we viewed each cluster and their pitch selection. However, it needed to be adjusted based on league averages for each different pitch. For example, four seam fastballs tend to be used more on average than others by starting pitchers, so we gave each different pitch a weight based on these averages in order to evenly compare them and provide more accurate clusterings. To achieve this we used ranks on each pitch to make comparisons.

R Version

```
library(tidyverse)
ranked_pitchers_by_outs <- aggregated_pitchers_by_outs %>%
  group_by(player_name, outs_when_up) %>%
  mutate(
    rank_FF = rank(weighted_FF, ties.method = "average"),
    rank_CH = rank(weighted_CH, ties.method = "average"),
    rank_SL = rank(weighted_SL, ties.method = "average"),
    rank_SI = rank(weighted_SI, ties.method = "average"),
    rank_CU = rank(weighted_CU, ties.method = "average"),
    rank_FC = rank(weighted_FC, ties.method = "average"),
    rank_FS = rank(weighted_FS, ties.method = "average"),
    rank_ST = rank(weighted_ST, ties.method = "average"),
    rank_KC = rank(weighted_KC, ties.method = "average"),
    rank_SV = rank(weighted_SV, ties.method = "average")
  ) %>%
  ungroup()
```

In the dataset above, each pitcher has a rank for each pitch.

Based off of the ranked pitches we were able to make more accurate clusters with the pitch selections weighted for general pitch usage. We did this in two ways: one ignoring the number of outs and just recording pitch selection in general and another by looking at pitch selection based on the number of outs for each cluster.

The chunk below sets up the pitching data for clustering based on pitch selection under given numbers of outs.

```
pitching_data_for_clustering <- aggregated_pitchers_by_outs %>%
  select(-player_name, -outs_when_up)
distance_matrix <- dist(pitching_data_for_clustering, method = "euclidean")
hc_pitchers <- hclust(distance_matrix, method = "ward.D2")
```

```r
clusters <- cutree(hc_pitchers, k = 4)

aggregated_pitchers_by_outs <- aggregated_pitchers_by_outs %>%
  mutate(cluster = clusters)
```

```r
ggplot(aggregated_pitchers_by_outs, aes(x = outs_when_up, y = weighted_FF, fill = factor(clus
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Fastball (FF) Selection by Cluster and Outs",
       x = "Outs", y = "Average Rank of Fastball (FF)") +
  theme_minimal()
```

In the graph above you can see the pitch selection remained fairly consistent with each number of outs. Cluster 3 threw more fastballs when there was 0 outs compared to 1 or 2 outs. Also, cluster 5 threw significantly more fastballs with 1 out than when they had 0 or 2 outs.

Python Version

```python
pitchers_grouped = grouped_pitches_by_outs.pivot(
    index=['player_name', 'outs_when_up'],
    columns='pitch_type',
    values='pitch_proportion'
).fillna(0).reset_index()
```

```
for col in pitchers_grouped.columns[2:]:  # Skip player_name and outs_when_up
    pitchers_grouped[f'weighted_{col}'] = (
        pitchers_grouped[col] * grouped_pitches_by_outs.groupby(['player_name', 'outs_when_up
    ) / grouped_pitches_by_outs.groupby(['player_name', 'outs_when_up'])['total_pitches_in_ou
```

```
ranked_pitchers_by_outs = pitchers_grouped.copy()
for col in [c for c in ranked_pitchers_by_outs.columns if c.startswith('weighted_')]:
    ranked_pitchers_by_outs[f'rank_{col[9:]}'] = ranked_pitchers_by_outs.groupby(['player_nam
```

```
ggplot(ranked_pitchers_by_outs, aes(x='outs_when_up', y='weighted_FF', color='player_name'))
    geom_line() + \
    labs(title="Weighted Fastball Usage by Outs", x="Outs", y="Weighted Fastball Usage")
```

```
<plotnine.ggplot.ggplot object at 0x0000025524D04D90>
```

Pitch Type Groups

This was a step in the right direction, however an issue came up when trying to visualize the data. Since each pitcher doesn't have every single pitch seen above in their arsenal, I needed to filter the data into three broader categories: fastballs, which included fourseams, sinkers, and cutters, offspeed pitches, which included changeups, knuckleballs, and splitters, and finally breaking balls, which included curveballs, sliders, knuckle curveballs, sweepers, and slurves. The code for this is shown below:

R Version

```
type_pitches <- filtered_pitchers %>%
  filter(!(pitch_type %in% c("CS", "PO"))) %>%
  mutate(pitch_group = case_when(
    pitch_type %in% c("FF", "SI", "FC") ~ "Fastball",
    pitch_type %in% c("CH", "FS", "KN") ~ "Offspeed",
    pitch_type %in% c("CU", "SL", "ST", "KC", "SV") ~ "Breaking_Ball"
  ))
```

With these new pitch groupings we once again clustered pitchers based on their arsenals with the 3 broad pitch types. Similarly, we cut the hierarchical clustering tree into 5 groups and viewed the pitch selection in each one by calculating the average proportion of pitches in each of the three groups.

```r
divided_pitches <- type_pitches %>%
  group_by(player_name, pitch_group) %>%
  summarize(count = n(), .groups = "drop") %>%
  group_by(player_name) %>%
  mutate(proportion = count / sum(count)) %>%
  select(-count) %>%
  pivot_wider(names_from = pitch_group, values_from = proportion, values_fill = 0)
```

```r
cluster_pitches <- divided_pitches %>%
  select(Breaking_Ball, Fastball, Offspeed)
```

```r
dist_matrix <- dist(cluster_pitches, method = "euclidean")
```

```r
hc <- hclust(dist_matrix, method = "ward.D2")
```

```r
clusters <- cutree(hc, k = 5)
cluster_pitches.eight <- cluster_pitches %>%
  ungroup() %>%
  mutate(cluster = clusters)
```

```r
pitch_selection <- cluster_pitches.eight %>%
  group_by(cluster) %>%
  summarize(
    avg_breaking_ball = mean(Breaking_Ball, na.rm = TRUE),
    avg_fastball = mean(Fastball, na.rm = TRUE),
    avg_offspeed = mean(Offspeed, na.rm = TRUE),
    count = n()
  )
```

The chunks below filter out "pitches" that were counted on caught stealings and pick-offs, we won't be using those in the data because they weren't actual pitches. Then it ranks each pitcher by pitch type usage with different numbers of outs.

```r
type_pitches_outs <- filtered_pitchers %>%
  filter(!(pitch_type %in% c("CS", "PO"))) %>%
  mutate(pitch_group = case_when(
    pitch_type %in% c("FF", "SI", "FC") ~ "Fastball",
    pitch_type %in% c("CH", "FS", "KN") ~ "Offspeed",
    pitch_type %in% c("CU", "SL", "ST", "KC", "SV") ~ "Breaking_Ball"
  ))
```

```r
type_pitches_outs_sum <- type_pitches_outs %>%
  group_by(player_name, outs_when_up, pitch_group) %>%
  summarize(count = n(), .groups = "drop") %>%
  group_by(player_name, outs_when_up) %>%
  mutate(proportion = count/sum(count)) %>%
  select(-count) %>%
  pivot_wider(names_from = pitch_group, values_from = proportion, values_fill = 0)
```

```r
ranks_outs <- type_pitches_outs_sum %>%
  mutate(last_name = str_extract(player_name, "^[^,]+")) %>%
  group_by(outs_when_up) %>%
  mutate(
    rank_breaking_ball = rank(-Breaking_Ball),
    rank_fastball = rank(-Fastball),
    rank_offspeed = rank(-Offspeed)
  ) %>%
  ungroup() %>%
  select(player_name, outs_when_up, rank_breaking_ball, rank_fastball, rank_offspeed, last_na
```

```r
ranks_outs_long <- ranks_outs %>%
  pivot_longer(cols = starts_with("rank"), names_to = "pitch_group", values_to = "rank") %>%
  mutate(pitch_group = gsub("rank_", "" ,pitch_group))
```

Python Version

```python
import numpy as np
# Remove non-pitch events
type_pitches_outs = filtered_pitchers[~filtered_pitchers["pitch_type"].isin(["CS", "PO"])].co

# Assign pitch group
type_pitches_outs["pitch_group"] = np.select(
    [
        type_pitches_outs["pitch_type"].isin(["FF", "SI", "FC"]),
        type_pitches_outs["pitch_type"].isin(["CH", "FS", "KN"]),
        type_pitches_outs["pitch_type"].isin(["CU", "SL", "ST", "KC", "SV"])
    ],
    ["Fastball", "Offspeed", "Breaking_Ball"],
    default="Unknown"
)
```

```
import re
import numpy as np
type_pitches_outs["pitch_group"] = np.select(
    [
        type_pitches_outs["pitch_type"].isin(["FF", "SI", "FC"]),
        type_pitches_outs["pitch_type"].isin(["CH", "FS", "KN"]),
        type_pitches_outs["pitch_type"].isin(["CU", "SL", "ST", "KC", "SV"])
    ],
    ["Fastball", "Offspeed", "Breaking_Ball"],
    default=None
)
```

```
# Remove non-pitch events
type_pitches_outs = filtered_pitchers[~filtered_pitchers["pitch_type"].isin(["CS", "PO"])].co

# Assign pitch group
type_pitches_outs["pitch_group"] = np.select(
    [
        type_pitches_outs["pitch_type"].isin(["FF", "SI", "FC"]),
        type_pitches_outs["pitch_type"].isin(["CH", "FS", "KN"]),
        type_pitches_outs["pitch_type"].isin(["CU", "SL", "ST", "KC", "SV"])
    ],
    ["Fastball", "Offspeed", "Breaking_Ball"],
    default="Unknown"
)

#  Summarize counts by player, outs, and pitch group
type_pitches_outs_sum = (
    type_pitches_outs
    .dropna(subset=["pitch_group"])
    .groupby(["player_name", "outs_when_up", "pitch_group"])
    .size()
    .reset_index(name="count")
)

# Compute proportions within each player_name + outs_when_up group
type_pitches_outs_sum = (
    type_pitches_outs_sum
    .groupby(["player_name", "outs_when_up"])
    .apply(lambda df: df.assign(proportion=df["count"] / df["count"].sum()))
    .reset_index(drop=True)
    .drop(columns="count")
```

```
    .pivot(index=["player_name", "outs_when_up"], columns="pitch_group", values="proportion")
    .fillna(0)
    .reset_index()
)
```

```
# Extract last name from "Lastname, Firstname"
type_pitches_outs_sum["last_name"] = type_pitches_outs_sum["player_name"].str.extract(r"^([^

ranks_outs = (
    type_pitches_outs_sum
    .copy()
    .groupby("outs_when_up")
    .apply(lambda df: df.assign(
        rank_breaking_ball = df["Breaking_Ball"].rank(ascending=False, method='min'),
        rank_fastball = df["Fastball"].rank(ascending=False, method='min'),
        rank_offspeed = df["Offspeed"].rank(ascending=False, method='min')
    ))
    .reset_index(drop=True)
)

ranks_outs = ranks_outs[[
    "player_name", "outs_when_up", "rank_breaking_ball", "rank_fastball", "rank_offspeed", "
]]
```

```
ranks_outs_long = (
    ranks_outs
    .melt(
        id_vars=["player_name", "outs_when_up", "last_name"],
        value_vars=["rank_breaking_ball", "rank_fastball", "rank_offspeed"],
        var_name="pitch_group",
        value_name="rank"
    )
)

ranks_outs_long["pitch_group"] = ranks_outs_long["pitch_group"].str.replace("rank_", "")
```

```
# Prepare data for clustering
pitching_data_for_clustering = aggregated_pitchers_by_outs.drop(columns = ["player_name", "ou
distance_matrix = pdist(pitching_data_for_clustering, metric = 'euclidean')
hc_pitchers = linkage(distance_matrix, method = 'ward')
```

```
# Cut tree into clusters
clusters = fcluster(hc_pitchers, 4, criterion = 'maxclust')
aggregated_pitchers_by_outs["cluster"] = clusters
```

```
# Remove non-pitch events
type_pitches = (
    filtered_pitchers[~filtered_pitchers["pitch_type"].isin(["CS", "PO"])]
    .copy()
)

# Assign pitch group
type_pitches["pitch_group"] = np.select(
    [
        type_pitches["pitch_type"].isin(["FF", "SI", "FC"]),
        type_pitches["pitch_type"].isin(["CH", "FS", "KN"]),
        type_pitches["pitch_type"].isin(["CU", "SL", "ST", "KC", "SV"])
    ],
    ["Fastball", "Offspeed", "Breaking_Ball"],
    default="Unknown"
)

# Proportion of each pitch group by pitcher
divided_pitches = (
    type_pitches[type_pitches["pitch_group"] != "Unknown"]
    .groupby(["player_name", "pitch_group"])
    .size()
    .reset_index(name="count")
    .groupby("player_name")
    .apply(lambda df: df.assign(proportion=df["count"] / df["count"].sum()))
    .reset_index(drop=True)
    .drop(columns="count")
    .pivot(index="player_name", columns="pitch_group", values="proportion")
    .fillna(0)
    .reset_index()
)

# Extract just the pitch proportions for clustering
cluster_pitches = divided_pitches[["Breaking_Ball", "Fastball", "Offspeed"]]

# Hierarchical clustering
distance_matrix = pdist(cluster_pitches, metric="euclidean")
hc = linkage(distance_matrix, method="ward")
```

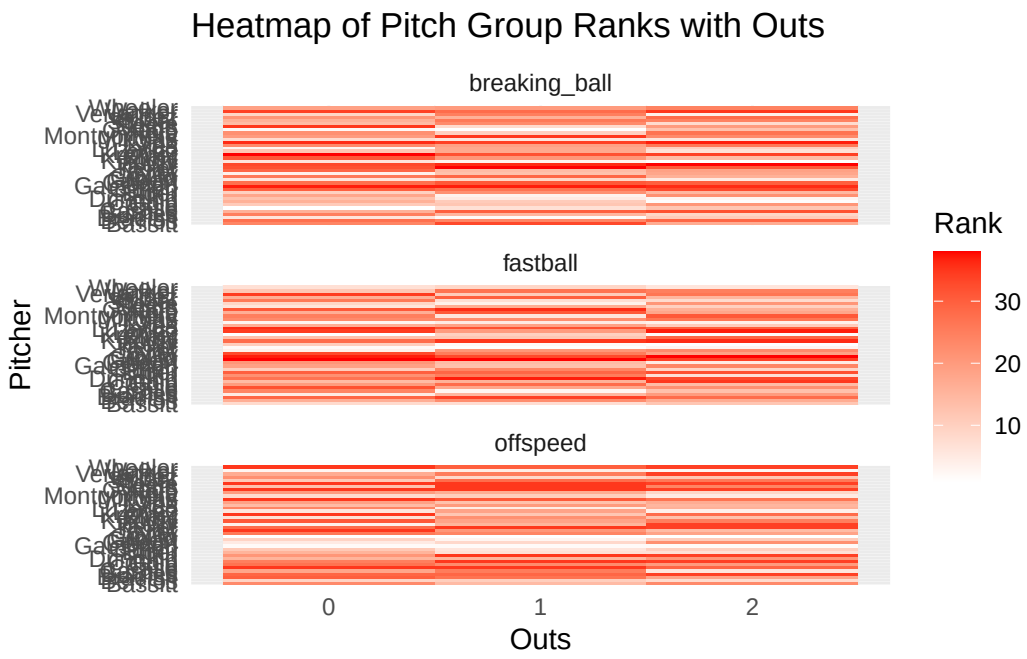
```python
# Cut tree into 5 clusters
clusters = fcluster(hc, 5, criterion="maxclust")

# Add cluster assignments back to full dataset
cluster_pitches_eight = divided_pitches.copy()
cluster_pitches_eight["cluster"] = clusters
```

To visualize these results we looked at plots and heatmaps to see differences between the different clusters.

Visualizations

R Version

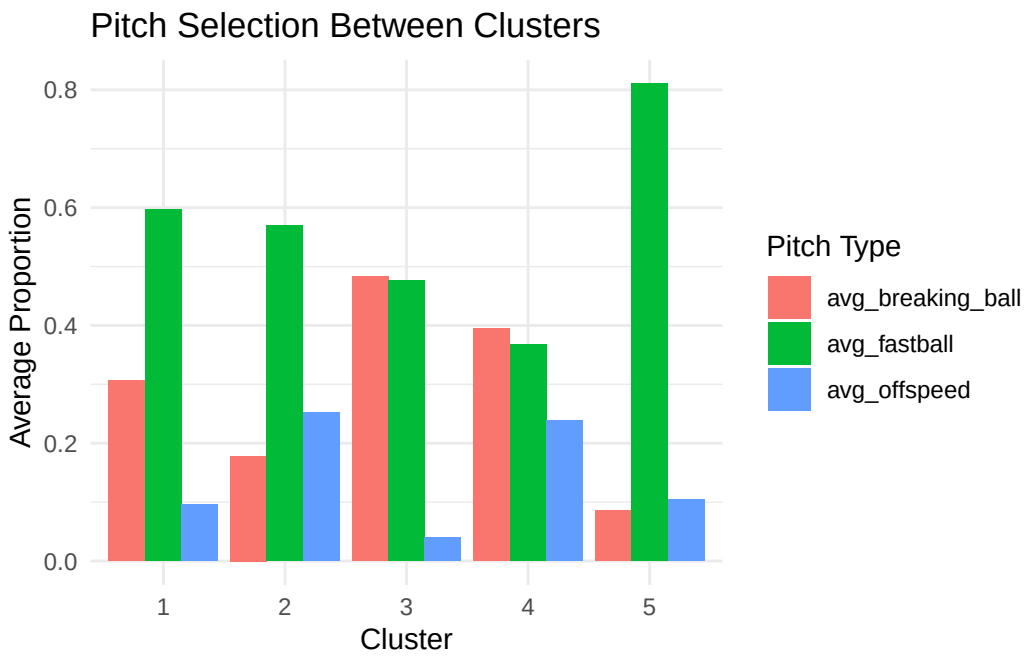
```r
ggplot(data = ranks_outs_long, mapping = aes(x = outs_when_up, y = last_name, fill = rank)) +
  geom_tile() +
  facet_wrap(~ pitch_group, ncol = 1) +
  scale_fill_gradient(low = "white", high = "red") +
  labs(
    x = "Outs",
    y = "Pitcher",
    fill = "Rank",
    title = "Heatmap of Pitch Group Ranks with Outs"
  ) +
  theme_minimal()
```




```r
pitch_selection <- cluster_pitches.eight %>%
  group_by(cluster) %>%
  summarize(
    avg_breaking_ball = mean(Breaking_Ball, na.rm = TRUE),
    avg_fastball = mean(Fastball, na.rm = TRUE),
    avg_offspeed = mean(Offspeed, na.rm = TRUE),
    count = n()
  )
```

```r
pitch_selection_plot <- pitch_selection %>%
  pivot_longer(cols = starts_with("avg"), names_to = "pitch_type", values_to = "proportion")
```

```r
ggplot(data = pitch_selection_plot, mapping = aes(x = factor(cluster), y = proportion, fill =
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Cluster", y = "Average Proportion", fill = "Pitch Type", title = "Pitch Selectio
  theme_minimal()
```



Python Version

```
# Group by cluster and compute average pitch proportions
pitch_selection = (
    cluster_pitches_eight
    .groupby("cluster")
    .agg(
        avg_breaking_ball=('Breaking_Ball', 'mean'),
        avg_fastball=('Fastball', 'mean'),
        avg_offspeed=('Offspeed', 'mean'),
        count=('cluster', 'count')
    )
    .reset_index()
)
```

```
# Fix last_name extraction
ranks_outs["last_name"] = ranks_outs["player_name"].str.extract(r"^([^,]+)")

# Reshape data to long format
ranks_outs_long = ranks_outs.melt(
    id_vars=["player_name", "outs_when_up", "last_name"],
    value_vars=["rank_breaking_ball", "rank_fastball", "rank_offspeed"],
```

```
    var_name="pitch_group",
    value_name="rank"
)

# Clean pitch_group column
ranks_outs_long["pitch_group"] = ranks_outs_long["pitch_group"].str.replace("rank_", "")

# Plot heatmap
heatmap_plot = (
    ggplot(ranks_outs_long, aes(x="outs_when_up", y="last_name", fill="rank")) +
    geom_tile() +
    facet_wrap("~pitch_group", ncol=1) +
    scale_fill_gradient(low="white", high="red") +
    labs(
        title="Heatmap of Pitch Group Ranks with Outs",
        x="Outs",
        y="Pitcher",
        fill="Rank"
    ) +
    theme_minimal()
)
```

```
pitch_selection = (
    cluster_pitches_eight
    .groupby("cluster")
    .agg(
        avg_breaking_ball=('Breaking_Ball', 'mean'),
        avg_fastball=('Fastball', 'mean'),
        avg_offspeed=('Offspeed', 'mean'),
        count=('cluster', 'count')
    )
    .reset_index()
)
```

```
# Pivot Longer
pitch_selection_plot = (
  pitch_selection
  .melt(id_vars = 'cluster', value_vars=['avg_breaking_ball', 'avg_fastball', 'avg_offspeed']
)
```

```
(
  ggplot(pitch_selection_plot, aes(x = 'factor(cluster)', y = 'proportion', fill = 'pitch_ty
  geom_bar(stat = 'identity', position = 'dodge') +
  labs(x = 'Cluster', y = 'Average Proportion', fill = 'Pitch Type',
    title = 'Pitch Selection Between Clusters') +
  theme_minimal()
)
```

```
<plotnine.ggplot.ggplot object at 0x0000025524D818A0>
```

Step 2: Situational Changes

After clustering the pitchers we moved on to see how they reacted in different ball-strike counts. Specifically, we were interested in how their pitch selection varied in different ball-strike counts.

Setting Up Counts Data

R Version

```
specific_counts <- type_pitches %>%
  #filter(count %in% c("0-2", "1-2")) %>%
  group_by(player_name, pitch_group, count) %>%
  summarize(count_specific = n(), .groups = "drop") %>%
  group_by(player_name, count) %>%
  mutate(proportion_specific = count_specific - sum(count_specific)) %>%
  select(player_name, pitch_group, count, proportion_specific)
```

To find this, we looked at the average change in proportion when the pitcher was ahead in the count (0-2 and 1-2) for each cluster. The purpose of this was to see which pitch was their go-to in a strikeout count and if it differed at all from their usual arsenal.

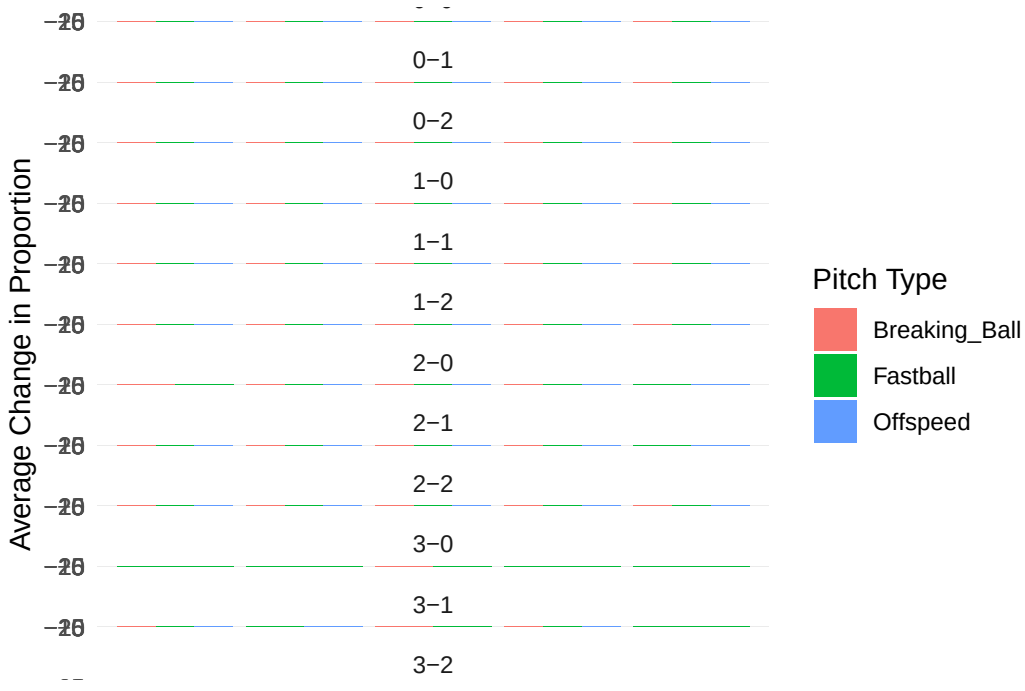
```
specific_counts <- type_pitches %>%
  #filter(count %in% c("0-2", "1-2")) %>%
  group_by(player_name, pitch_group, count) %>%
  summarize(count_specific = n(), .groups = "drop") %>%
  group_by(player_name, count) %>%
  mutate(proportion_specific = count_specific - sum(count_specific)) %>%
  select(player_name, pitch_group, count, proportion_specific)
```

```r
general_selection <- divided_pitches %>%
  pivot_longer(cols = c(Fastball, Breaking_Ball, Offspeed),
               names_to = "pitch_group", values_to = "proportion_general")
```

```r
comparison <- specific_counts %>%
  left_join(general_selection, by = c("player_name", "pitch_group")) %>%
  mutate(proportion_difference = proportion_specific - proportion_general)
```

```r
comparison_cluster <- comparison %>%
  left_join(cluster_pitches.eight %>% select(player_name, cluster), by = "player_name") %>%
  group_by(cluster, count, pitch_group) %>%
  summarize(avg_proportion_difference = mean(proportion_difference, na.rm = TRUE), .groups =
```

```r
ggplot(data = comparison_cluster, aes(x = factor(cluster), y = avg_proportion_difference, fi
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~ count, ncol = 1) +
  labs(x = "Cluster", y = "Average Change in Proportion", fill = "Pitch Type",
       title = "Change in Pitch Selection in 0-2 and 1-2 Counts by Cluster") +
  theme_minimal()
```



Python Version

```
specific_counts = (
    type_pitches
    .groupby(['player_name', 'pitch_group', 'count'])
    .size()
    .reset_index(name='count_specific')
    .groupby(['player_name', 'count'])
    .apply(lambda x: x.assign(proportion_specific = x['count_specific'] / x['count_specific']
    .reset_index(drop=True)
)
```

```
general_selection = (
  divided_pitches
  .melt(id_vars='player_name',
        value_vars = ['Fastball', 'Breaking_Ball', 'Offspeed'],
        var_name = 'pitch_group',
        value_name = 'proportion_general')
)
```

```
comparison = (
  specific_counts
  .merge(general_selection, on=['player_name', 'pitch_group'], how = 'left')
  .assign(proportion_difference=lambda df: df['proportion_specific'] - df['proportion_general
)
```

```
comparison_cluster = (
  comparison
  .merge(cluster_pitches_eight[['player_name', 'cluster']], on = 'player_name', how = 'left'
  .groupby(['cluster', 'count', 'pitch_group'])
  .agg(avg_proportion_difference=('proportion_difference', 'mean'))
  .reset_index()
)
```

```
(
  ggplot(comparison_cluster, aes(x = 'factor(cluster)', y = 'avg_proportion_difference', fil
  geom_bar(stat = 'identity', position = 'dodge') +
  facet_wrap('~count', ncol = 1) +
  labs(x = 'Cluster', y = 'Average Change in Proportion', fill = 'Pitch Type',
        title = 'Change in Pitch Selection in 0-2 and 1-2 Counts by Cluster') +
  theme_minimal()
)
```

```
<plotnine.ggplot.ggplot object at 0x0000025524FB47F0>
```

Standardization

After visualizing the proportions, we chose to standardize them using z-scores, which allowed us to make more effective comparisons. We then compared the standardized changes in pitch usages both within and between the different clusters.

The code below reformats the data and computes the average change in pitch usage by pitch type and cluster.

R Version

```
count_prop <- type_pitches %>%
#  filter(count %in% c("0-2", "1-2")) %>%
  group_by(player_name, count) %>%
  mutate(total_pitches = n()) %>%
  group_by(player_name, pitch_group, count) %>%
  summarise(proportion = n() / unique(total_pitches), .groups = 'drop')
```

```
# adding cluster column
comparison <- comparison %>%
  left_join(cluster_pitches.eight %>% select(player_name, cluster), by = "player_name")
```

```
count_diff <- count_prop %>%
  inner_join(comparison, by = c("player_name", "count", "pitch_group")) %>%
  mutate(difference = proportion_general - proportion)
```

```
count_diff <- count_diff %>%
  select(-proportion_specific, -proportion_difference)
```

```
avg_diff <- count_diff %>%
  group_by(player_name, count, cluster) %>%
  summarise(
    change_fastball = mean(difference[pitch_group == "Fastball"], na.rm = TRUE) %>% replace_
    change_breaking_ball = mean(difference[pitch_group == "Breaking_Ball"], na.rm = TRUE) %>%
    change_offspeed = mean(difference[pitch_group == "Offspeed"], na.rm = TRUE) %>% replace_
  ) %>%
  ungroup()
```

Python Version

```
# Group by player_name and count to compute total pitches per group
type_pitches["total_pitches"] = (
    type_pitches
    .groupby(["player_name", "count"])["pitch_type"]
    .transform("count")
)

# Now group by player_name, pitch_group, and count to calculate proportions
count_prop = (
    type_pitches
    .groupby(["player_name", "pitch_group", "count"])
    .size()
    .reset_index(name="pitch_count")
)

# Merge total_pitches back in if not already included
count_prop = count_prop.merge(
    type_pitches[["player_name", "count", "total_pitches"]].drop_duplicates(),
    on=["player_name", "count"],
    how="left"
)

# Calculate proportion
count_prop["proportion"] = count_prop["pitch_count"] / count_prop["total_pitches"]
```

```
from IPython.display import display
display(type_pitches)
```

	pitch_type	game_date	release_speed	...	count	pitch_group	total_pitches
0	SL	10/1/2023	82.9	...	2-1	Breaking_Ball	13
1	SL	10/1/2023	83.7	...	2-0	Breaking_Ball	10
2	CH	10/1/2023	82.0	...	1-0	Offspeed	25
3	CH	10/1/2023	82.9	...	0-0	Offspeed	44
4	CU	10/1/2023	77.3	...	2-2	Breaking_Ball	17
...	...	...	...	...	...	...	...
24959	SI	9/26/2023	90.9	...	0-0	Fastball	20
24979	FF	9/26/2023	94.5	...	1-1	Fastball	15
24987	SI	9/26/2023	90.4	...	1-1	Fastball	9
24994	SL	9/26/2023	85.4	...	0-0	Breaking_Ball	20
24997	SI	9/26/2023	93.6	...	0-0	Fastball	21

[3737 rows x 97 columns]

```
type_pitches.columns
```

```
Index(['pitch_type', 'game_date', 'release_speed', 'release_pos_x',
       'release_pos_z', 'player_name', 'batter', 'pitcher', 'events',
       'description', 'spin_dir', 'spin_rate_deprecated',
       'break_angle_deprecated', 'break_length_deprecated', 'zone', 'des',
       'game_type', 'stand', 'p_throws', 'home_team', 'away_team', 'type',
       'hit_location', 'bb_type', 'balls', 'strikes', 'game_year', 'pfx_x',
       'pfx_z', 'plate_x', 'plate_z', 'on_3b', 'on_2b', 'on_1b',
       'outs_when_up', 'inning', 'inning_topbot', 'hc_x', 'hc_y',
       'tfs_deprecated', 'tfs_zulu_deprecated', 'fielder_2', 'umpire', 'sv_id',
       'vx0', 'vy0', 'vz0', 'ax', 'ay', 'az', 'sz_top', 'sz_bot',
       'hit_distance_sc', 'launch_speed', 'launch_angle', 'effective_speed',
       'release_spin_rate', 'release_extension', 'game_pk', 'pitcher.1',
       'fielder_2.1', 'fielder_3', 'fielder_4', 'fielder_5', 'fielder_6',
       'fielder_7', 'fielder_8', 'fielder_9', 'release_pos_y',
       'estimated_ba_using_speedangle', 'estimated_woba_using_speedangle',
       'woba_value', 'woba_denom', 'babip_value', 'iso_value',
       'launch_speed_angle', 'at_bat_number', 'pitch_number', 'pitch_name',
       'home_score', 'away_score', 'bat_score', 'fld_score', 'post_away_score',
       'post_home_score', 'post_bat_score', 'post_fld_score',
       'if_fielding_alignment', 'of_fielding_alignment', 'spin_axis',
       'delta_home_win_exp', 'delta_run_exp', 'bat_speed', 'swing_length',
       'count', 'pitch_group', 'total_pitches'],
      dtype='object')
```

```
count_prop = count_prop.merge(
    aggregated_pitchers_by_outs[['player_name', 'cluster']],
    on='player_name',
    how='left'
)
```

```
count_diff = (
    count_prop
    .merge(comparison, on=["player_name", "count", "pitch_group"], how="inner")
    .assign(difference = lambda df: df["proportion_general"] - df["proportion"])
)
count_diff = count_diff.drop(columns=["proportion_specific", "proportion_difference"], errors
```

```
avg_diff = (
    count_diff.groupby(['player_name', 'count', 'cluster'])
    .apply(lambda df: pd.Series({
        'change_fastball': df.loc[df['pitch_group'] == 'Fastball', 'difference'].mean() or 0
        'change_breaking_ball': df.loc[df['pitch_group'] == 'Breaking_Ball', 'difference'].me
        'change_offspeed': df.loc[df['pitch_group'] == 'Offspeed', 'difference'].mean() or 0
    }))
    .reset_index()
)
```

```
from scipy.stats import zscore

# Standardize pitch type changes across all pitchers
zscore_diff = avg_diff.copy()
for col in ['change_fastball', 'change_breaking_ball', 'change_offspeed']:
    zscore_diff[f'z_{col}'] = zscore(zscore_diff[col])
```

ShinyApp/Plotly

Following this and reformatting the data, we made a small shiny app to visualize the results. The shiny app allows you to visualize the different pitch selection for each cluster under different counts. This includes seeing how the pitchers changed their fastball, breaking ball, and offspeed pitch selection. In addition, it includes a 3D plot to visualize the different changes. Since there are three different pitch groupings, a 3D visual was neccesary to view the data and took us one step closer to look into modeling.

R Version

```
library(plotly)
library(shiny)
ui <- fluidPage(
  titlePanel("3D Pitch Type Change by Pitcher"),
  sidebarLayout(
    sidebarPanel(
      selectInput("count", "Select Count", choices = unique(avg_diff$count)),
      selectInput("cluster", "Select Cluster", choices = unique(avg_diff$cluster))
    ),
    mainPanel(
      plotlyOutput("pitch3DPlot")
    )
```

```
  )
)

# Define server logic
server <- function(input, output) {
  filtered_data <- reactive({
    avg_diff %>%
      filter(count == input$count, cluster == input$cluster)
  })

  output$pitch3DPlot <- renderPlotly({
    plot_ly(
      data = filtered_data(),
      x = ~change_fastball,
      y = ~change_breaking_ball,
      z = ~change_offspeed,
      type = "scatter3d",
      mode = "markers",
      text = ~player_name,
      marker = list(size = 5)
    ) %>%
      layout(
        scene = list(
          xaxis = list(title = "Change in Fastball"),
          yaxis = list(title = "Change in Breaking Ball"),
          zaxis = list(title = "Change in Offspeed")
        ),
        title = "3D Plot of Pitch Type Changes by Pitcher"
      )
  })
}

shinyApp(ui = ui, server = server)
```

Python Version

```
# Plotly version
import plotly.express as px
import plotly.graph_objects as go
import ipywidgets as widgets
from IPython.display import display
```

```
count_dropdown = widgets.Dropdown(options=avg_diff['count'].unique(), description = 'Count:')

# Dropdown for cluster
cluster_dropdown = widgets.Dropdown(options=avg_diff['cluster'].unique(), description = 'Clus
```

```
def update_plot(count_val, cluster_val):
    filtered = avg_diff[(avg_diff['count'] == count_val) & (avg_diff['cluster'] == cluster_va

    fig = go.Figure(data = [go.Scatter3d(
        x = filtered['change_fastball'],
        y = filtered['change_breaking_ball'],
        z = filtered['change_offspeed'],
        mode = 'markers',
        text = filtered['player_name'],
        marker = dict(size = 5)  # This is where the comma was missing
    )])

    fig.update_layout(
        title = '3D Plot of Pitch Type Changes by Pitcher',
        scene = dict(
            xaxis_title = 'Change in Fastball',
            yaxis_title = 'Change in Breaking Ball',
            zaxis_title = 'Change in Offspeed'
        )
    )
    fig.show()

#widgets.interactive(update_plot, count_val=count_dropdown, cluster_val=cluster_dropdown)
```

Step 3: Modeling

The next step in the process was to test out some models to see if we could predict the amount a pitcher varies from their average selection in different situations. Also, we included statistical measures as explanatory variables to see if their is a difference in variation between more and less successful pitchers. To calculate the variation we used the sum of the changes in the squared pitch groups (fastballs, breaking balls, and offspeed) and the absolute sum of the changes by taking the absolute value of each change for each type of pitch. The code for which can be found below:

Loading Pitching Data

R Version

```r
pitching23 <- read_csv("pitching23.csv")
```

```r
era_pitchers <- pitching23 %>%
  separate(Player, into = c("Firstname", "Lastname"), sep = " ", extra = "merge") %>%
  mutate(Player = paste(Lastname, Firstname, sep = ", "),
         ERA = ERA...3) %>%
  select(Player, ERA)
```

```r
# Perform the join
avg_diff <- avg_diff %>%
  left_join(era_pitchers, by = c("player_name" = "Player"))
```

```r
# Calculate SSE and absolute sum of changes for each pitcher
error_df <- avg_diff %>%
  group_by(player_name, count) %>%
  summarize(
    SSE = sum((change_fastball)^2 + (change_breaking_ball)^2 + (change_offspeed)^2),
    rSSE = sqrt(SSE),
    abs_sum = sum(abs(change_fastball) + abs(change_breaking_ball) + abs(change_offspeed)),
    ERA = first(ERA)  # Keep ERA the same for all counts of a player
  ) %>%
  ungroup()
```

Python Version

```python
# Load in data
pitching23 = pd.read_csv("pitching23.csv")
```

```python
pitching23[['Firstname', 'Lastname']] = pitching23['Player'].str.split(' ', n=1, expand=True)
pitching23['Player'] = pitching23['Lastname'] + ', ' + pitching23['Firstname']
era_pitchers = pitching23[['Player', 'ERA']]
```

```python
avg_diff = avg_diff.merge(era_pitchers, left_on = 'player_name', right_on = 'Player', how =
avg_diff.drop(columns=['Player'], inplace = True)
```

```
def compute_metrics(group):
    return pd.Series({
        'SSE': ((group['change_fastball'])**2 +
                (group['change_breaking_ball'])**2 +
                (group['change_offspeed'])**2).sum(),
        'abs_sum': (group['change_fastball'].abs() +
                    group['change_breaking_ball'].abs() +
                    group['change_offspeed'].abs()).sum(),
        'ERA': group['ERA'].iloc[0]  # or .mean() if more appropriate
    })

error_df = avg_diff.groupby(['player_name', 'count']).apply(compute_metrics).reset_index()
error_df['rSSE'] = error_df['SSE']**0.5
```

More Modeling

For the actual modeling we used statistics like ERA, different counts, strikeouts, WHIP, FIP, strikeout-to-walk ratio, innings pitched, and batters faced to predict SSE and the absolute sum of changes. We attempted various model types including linear and log models to find the best fit or at least look for significant explanatory variables.

R Version

```
so_pitchers <- pitching23 %>%
  separate(Player, into = c("Firstname", "Lastname"), sep = " ", extra = "merge") %>%
  mutate(Player = paste(Lastname, Firstname, sep = ", "),
         ERA = ERA...3) %>%
  select(Player, SO)
```

```
# Perform the join
avg_diff_so <- avg_diff %>%
  left_join(so_pitchers, by = c("player_name" = "Player"))
```

```
# Calculate SSE and absolute sum of changes for each pitcher
error_df_so <- avg_diff_so %>%
  group_by(player_name, count) %>%
  summarize(
    SSE = sqrt(sum((change_fastball)^2 + (change_breaking_ball)^2 + (change_offspeed)^2)),
    abs_sum = sum(abs(change_fastball) + abs(change_breaking_ball) + abs(change_offspeed)),
    SO = first(SO)
```

```
  ) %>%
  ungroup()
```

```
error_df_so_adj <- error_df_so %>%
  mutate(
    # Split 'count' into balls and strikes
    balls = as.numeric(sub("-.*", "", count)),
    strikes = as.numeric(sub(".*-", "", count)),
    adjusted_count = strikes - balls
  )
```

```
pitching23_cut <- pitching23 %>%
  mutate(Player = sub("^(\\S+)\\s+(.*)$", "\\2, \\1", Player)) %>%
  select(Player, FIP, WHIP, `SO/BB`, BF, IP) # Select only the desired columns
```

```
stat_df <- error_df_so_adj %>%
  left_join(pitching23_cut, by = c("player_name" = "Player"))
```

```
mod_int3 <- lm(log(SSE) ~ as.factor(adjusted_count) + FIP + `SO/BB` + IP, data = stat_df)
summary(mod_int3)
```

```
Call:
lm(formula = log(SSE) ~ as.factor(adjusted_count) + FIP + `SO/BB` +
    IP, data = stat_df)

Residuals:
    Min      1Q  Median      3Q     Max
-2.9505 -0.3716  0.1443  0.4557  1.6057

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 -0.248894   0.672291  -0.370 0.711408
as.factor(adjusted_count)-2 -0.163465   0.177677  -0.920 0.358096
as.factor(adjusted_count)-1 -0.602318   0.167886  -3.588 0.000373 ***
as.factor(adjusted_count)0  -0.934577   0.167771  -5.571 4.55e-08 ***
as.factor(adjusted_count)1  -0.965611   0.174418  -5.536 5.46e-08 ***
as.factor(adjusted_count)2  -0.443546   0.192990  -2.298 0.022037 *
FIP                         -0.067337   0.058864  -1.144 0.253301
`SO/BB`                      0.037960   0.030507   1.244 0.214081
```

```
IP                          -0.002998   0.003085  -0.972 0.331696
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7201 on 419 degrees of freedom
Multiple R-squared:  0.1765,    Adjusted R-squared:  0.1608
F-statistic: 11.23 on 8 and 419 DF,  p-value: 1.988e-14
```

Python Version

```python
# Start from the original DataFrame so we retain all columns
so_pitchers = pitching23.copy()

# Split 'Player' column into 'Firstname' and 'Lastname'
so_pitchers[['Firstname', 'Lastname']] = so_pitchers['Player'].str.split(' ', n=1, expand=Tru

# Create new 'Player' name in 'Lastname, Firstname' format
so_pitchers['Player'] = so_pitchers['Lastname'] + ', ' + so_pitchers['Firstname']


# Now select the relevant columns
so_pitchers = so_pitchers[['Player', 'SO']]
```

```python
# Perform the join
avg_diff_so = avg_diff.merge(so_pitchers, left_on='player_name', right_on='Player', how='left
```

```python
# Calculate SSE and absolute sum of changes for each pitcher
error_df_so = avg_diff_so.groupby(['player_name', 'count']).apply(
    lambda x: pd.Series({
        'SSE': (x['change_fastball']**2 + x['change_breaking_ball']**2 + x['change_offspeed']
        'abs_sum': (abs(x['change_fastball']) + abs(x['change_breaking_ball']) + abs(x['chang
        'SO': x['SO'].iloc[0]  # Assuming 'SO' is consistent for each player_name and count
    })
).reset_index()
```

```python
def compute_metrics_so(group):
    return pd.Series({
        'SSE': ((group['change_fastball']**2 +
                 group['change_breaking_ball']**2 +
                 group['change_offspeed']**2).sum())**0.5,
        'abs_sum': (group['change_fastball'].abs() +
```

```
                    group['change_breaking_ball'].abs() +
                    group['change_offspeed'].abs()).sum(),
        'SO': group['SO'].iloc[0]  # take the first value, or use .mean() if appropriate
    })

error_df_so = (
    avg_diff_so
    .groupby(['player_name', 'count'])
    .apply(compute_metrics_so)
    .reset_index()
)
```

```
# Adjusted Counts
error_df_so['balls'] = error_df_so['count'].str.extract(r'(\d)-')[0].astype(int)
error_df_so['strikes'] = error_df_so['count'].str.extract(r'(\d)-')[0].astype(int)
error_df_so['adjusted_count'] = error_df_so['strikes'] - error_df_so['balls']
```

```
pitching23_cut = pitching23[['Player', 'FIP', 'WHIP', 'SO/BB', 'BF', 'IP']].copy()
stat_df = error_df_so.merge(pitching23_cut, left_on = 'player_name', right_on = 'Player', hou
```

```
stat_df['log_SSE'] = np.log(stat_df['SSE'] + 1e-6)
stat_df = stat_df.replace([np.inf, -np.inf], np.nan).dropna(subset=['log_SSE'])
```

```
# Fitting the Model
import statsmodels.formula.api as smf
import numpy as np

# Replace 0 SSE values with a small constant to avoid log(0)
stat_df['log_SSE'] = np.log(stat_df['SSE'].replace(0, 1e-6))

# Remove rows where log_SSE is still inf or NaN
stat_df = stat_df.replace([np.inf, -np.inf], np.nan)
stat_df = stat_df.dropna(subset=['log_SSE'])

# Convert count to category
stat_df['adjusted_count'] = stat_df['adjusted_count'].astype('category')

import statsmodels.formula.api as smf
mod_int3 = smf.ols('log_SSE ~ C(adjusted_count) + FIP + Q("SO/BB") + IP', data=stat_df).fit()

# View results
print(mod_int3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                log_SSE   R-squared:                       0.009
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     1.264
Date:                Thu, 08 May 2025   Prob (F-statistic):              0.286
Time:                        22:32:55   Log-Likelihood:                -1369.3
No. Observations:                 428   AIC:                             2747.
Df Residuals:                     424   BIC:                             2763.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -17.3678      5.434     -3.196      0.001     -28.050      -6.686
FIP            0.4344      0.487      0.892      0.373      -0.523       1.392
Q("SO/BB")     0.3308      0.252      1.310      0.191      -0.165       0.827
IP             0.0298      0.026      1.167      0.244      -0.020       0.080
==============================================================================
Omnibus:                     2220.127   Durbin-Watson:                   1.511
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               68.279
Skew:                           0.413   Prob(JB):                     1.49e-15
Kurtosis:                       1.226   Cond. No.                     3.48e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.48e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In this original model we see the adjusted count of -1, 0, 1, and 2 all being significant, suggesting that in those situations where the pitcher is either even, down 1 ball, up 1 strike, or up 2 strikes, it has an impact on how much they are deviating. The negative coefficients suggest that there is less deviation from their original arsenal in these counts. With a coefficient of -0.966, being up 1 strike has the strongest impact. However, statistical performance measures such as FIP, strikeout-to-walk ratio, and innings pitched don't seem to have a significant effect. Meaning you can't use them to predict a pitcher's deviance from their usual pitch selection.

Conclusion

This project aimed to uncover patterns in pitch selection among pitchers by clustering them based on their usage of fastballs, breaking balls, and offspeed pitches. By applying hierarchical

clustering techniques and visualizing the results through various visualizations, we identified distinct groups of pitchers with similar tendencies. These clusters highlighted each pitcher's approach.

We further explored these situational adaptations by examining pitch selection changes in key ball-strike counts (e.g., 0-2 and 1-2) compared to general tendencies. By standardizing these changes using z-scores, we enabled fairer comparisons across players and clusters.

Overall, this analysis provides a deeper understanding of how pitchers strategize, both generally and situationally. It has potential applications in scouting, player development, and in-game decision-making. Future extensions of this work could include exploring why the pitchers selection were changing in certain counts and including relief pitchers in high stress situations.