Predicting MLB Pitches

April 14, 2024

Goal: The pitches thrown during Sepetmeber 2019 in the MLB were tested to see if pitch, player, and situational metrics could predict what types of pitches were about to be thrown, potentially making it easier for hitters to react to different pitchers.

Model Accuracy: 80%

The model determined that the varaibles release speed, release spin rate, pitcher name, and release position were the most important in predicting what pitch is coming.

Figure 1 below shows the relationship between release speed and release spin rate for the nine different pitches thrown. Pitches such as a 4-seam fastball and sinker have high effective speeds, while sliders and curveballs have the highest release spin rates.

[19]:
```
#importing data/libraries, changing 'pitch type' names, and adding dummy␣
 ↪variables
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import zipfile
from sklearn.tree import DecisionTreeClassifier
from mlxtend.plotting import plot_decision_regions

filename="assets/baseball_svm_data.zip"
df=pd.read_csv(zipfile.ZipFile(filename).open("reg_Sep2019.csv"))
df=df.drop(df[df["pitch_type"]=='EP'].index)
pd.set_option('display.max_columns', None)
df['Pitcher_run_diff'] = df['fld_score'] - df['bat_score']
df['Pitcher_ahead_count'] = df['strikes'] - df['balls']
df['pitcher_match_up'] = df.apply(lambda row: 1 if row['p_throws'] ==␣
 ↪row['stand'] else 2, axis=1)
df['if_defense'] = df['if_fielding_alignment'].map({'Standard': 1, 'Infield␣
 ↪shift': 2, 'Strategic': 3})
df['of_defense'] = df['of_fielding_alignment'].map({'Standard': 1, '4th␣
 ↪outfielder': 2, 'Strategic': 3})
df['on_base'] = df.apply(lambda row: 1 if not pd.isna(row['on_1b']) or not pd.
 ↪isna(row['on_2b']) or not pd.isna(row['on_3b']) else 0, axis=1)
```

```
pitch_type_mapping = {
    'CH': 'CHG',
    'CU': 'CUR',
    'FC': 'CUT',
    'FF': '4SF',
    'FS': 'SPL',
    'FT': '2SF',
    'KC': 'KCV',
    'SI': 'SNK',
    'SL': 'SLD'
}


df['pitch_type'] = df['pitch_type'].replace(pitch_type_mapping)
```

[20]:
```
#add colors
color_list=list(mcolors.TABLEAU_COLORS.keys())
df2=df.groupby("pitch_type").apply(lambda x: x.assign(color=color_list.pop())).
 ↪reset_index(drop=True)

# create scatter plot
fig, ax = plt.subplots(figsize=(12, 6))
for key, group in df2.groupby('pitch_type'):
    group.plot.scatter("release_speed", "release_spin_rate", ax=ax, s=15,␣
 ↪label=key, color=group['color'])

# legend
ax.legend(title='Pitch Type', title_fontsize='large', fontsize='large')


# title and labels
ax.set_title("Release Spin vs. Release Speed", fontsize='x-large')
ax.set_xlabel("Effective Speed", fontsize='large')
ax.set_ylabel("Release Spin Rate", fontsize='large')
plt.show()
```

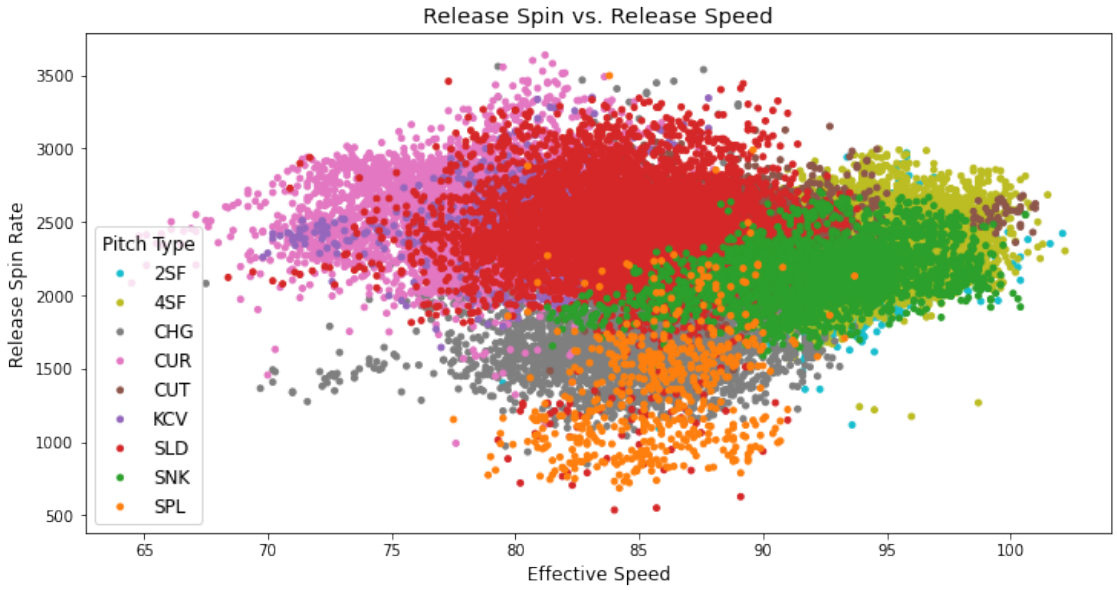


Figure 1 (Release Speed vs. Release Spin Rate)

Figure 2 below shows the percent of players that throw each pitch, its average spin rate, and release speed. About 90% of player have a fastball in their arsenal, while only about 7% throw the knuckle curve and splitter.

[21]:
```
#Making datatable
df_cleaned = df.dropna(subset=['release_spin_rate', 'release_speed',␣
 ↪'player_name'])

# Grouping by 'pitch type', calculating percentages and means
pitch_stats = df_cleaned.groupby('pitch_type').agg({
    'player_name': lambda x: x.nunique() / 540,
    'release_spin_rate': 'mean',
    'release_speed': 'mean'
}).reset_index()

# Renaming columns
pitch_stats.columns = ['Pitch Type', 'Percent of Players', 'AVG Spin Rate',␣
 ↪'AVG Release Speed']

# Sorting the DataFrame by the specified columns
pitch_stats.sort_values(by=[  'Percent of Players','AVG Spin Rate','AVG Release␣
 ↪Speed'], ascending=[False, False, False], inplace=True)

print(pitch_stats)
```

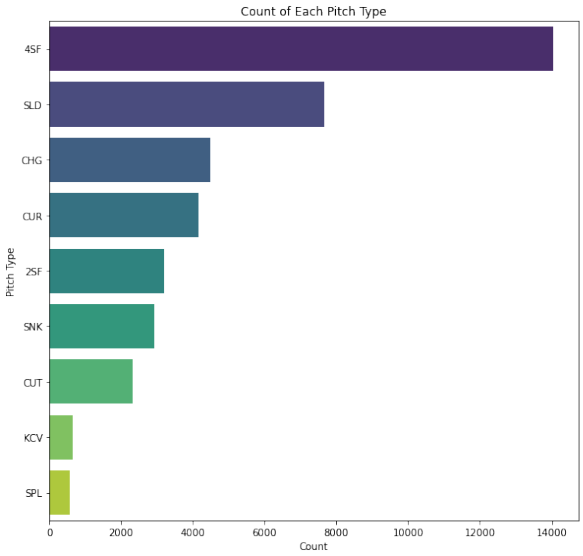
```
  Pitch Type  Percent of Players  AVG Spin Rate  AVG Release Speed
```

1	4SF	0.890741	2299.025716	93.551346
6	SLD	0.720370	2432.308296	84.796059
2	CHG	0.635185	1789.750627	84.800479
3	CUR	0.500000	2540.236294	78.759670
0	2SF	0.351852	2170.927233	92.710837
7	SNK	0.220370	2137.761204	92.280260
4	CUT	0.207407	2382.138736	88.672665
5	KCV	0.072222	2509.304147	80.346697
8	SPL	0.070370	1361.617591	85.725813

Figure 2 (Pitches Sorted by Percent of Players That Throw it, as Well as Their Average Spin Rate and Release Speed)

Figure 3 below shows the count of each pitch thrown in Septmeber 2019. About 14,000 fastballs were thrown, while lesss than 1,000 splitters and knuckle curves were thrown.

[22]:
```
import seaborn as sns
# Grouping by 'pitch type' and counting
counts = df.groupby("pitch_type").apply(lambda z: len(z)).
 ↪reset_index(name='count').sort_values(by='count', ascending=False)

# Creating subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 8))

# bar plot
sns.barplot(x='count', y='pitch_type', data=counts, ax=axes[0],␣
 ↪palette='viridis', orient='h')
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Pitch Type')
axes[0].set_title('Count of Each Pitch Type')

#  table
axes[1].axis('off')
table = axes[1].table(cellText=counts.values, colLabels=counts.columns,␣
 ↪loc='center', cellLoc='center')
table.auto_set_font_size(False)
table.set_fontsize(16)

# column width
table.auto_set_column_width([0, 1])
table.scale(3.5, 3)

plt.tight_layout()
plt.show()
```



pitch_type	count
4SF	14039
SLD	7656
CHG	4485
CUR	4141
2SF	3203
SNK	2924
CUT	2309
KCV	656
SPL	575

Figure 3 (Pitches Sorted by Count)

MODEL TESTING

[23]:
```
pitch_metrics=['release_spin_rate','release_extension','release_pos_y','release_pos_x','releas

player_metrics=['player_name']

game_details=['outs_when_up','inning','Pitcher_run_diff','Pitcher_ahead_count','pitcher_match_
```

[24]:
```
#Creating new dataframe based on selected variables
df=df[[*pitch_metrics, *player_metrics, *game_details, "pitch_type"]]

# dropping NaN
df=df.dropna(subset=["pitch_type"])

#factorize on the player name, since we need numeric values to test

df['player_name']=df['player_name'].factorize()[0]
df=df.fillna(df.mean())

# Convert specified columns to objects
df['pitcher_match_up'] = df['pitcher_match_up'].astype(str)
df['if_defense'] = df['if_defense'].astype(str)
df['of_defense'] = df['of_defense'].astype(str)
df['on_base'] = df['on_base'].astype(str)
```

[25]:
```
# Sample 80% of the data for training
train_df = df.sample(frac=0.8, random_state=42)
```

```
# The remaining 20% will be used for testing
test_df = df.drop(train_df.index)
```

[26]:
```
X_train=train_df[train_df.columns.drop("pitch_type")]
y_train=train_df["pitch_type"]
X_test=test_df[test_df.columns.drop("pitch_type")]
y_test=test_df["pitch_type"]
```

[27]:
```
#GBM
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_validate


gbm = GradientBoostingClassifier(n_estimators=60,max_depth=5,random_state=42)

# Fit the model to the training data
gresults=cross_validate(gbm,X_train,y_train,cv=5,scoring='accuracy')
#Fit to testing data
gbm.fit(X_train,y_train)
gbmtest = gbm.score(X_test,y_test)
```

[28]:
```
#TREE
from sklearn.tree import DecisionTreeClassifier
from mlxtend.plotting import plot_decision_regions

clfs={}
clfs["dt_3"]=DecisionTreeClassifier(max_depth=5, random_state=1337)
clfs["dt_4"]=DecisionTreeClassifier(max_depth=6, random_state=1337)
for label, model in clfs.items():
    # training
    results=cross_validate(model,X_train,y_train,cv=5,scoring='accuracy')
    cv_acc=np.mean(results['test_score'])
    # testing
    val_acc=model.fit(X_test,y_test).score(X_test,y_test)
```

[29]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate

rf = RandomForestClassifier(n_estimators=30, max_depth=4, random_state=42)

#training
rf_results = cross_validate(rf, X_train, y_train, cv=5, scoring='accuracy')
#testing
rf.fit(X_train,y_train)
rfTest= rf.score(X_test,y_test)
```

[30]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of␣
 ↪neighbors as per your requirement

#training
kresults=cross_validate(knn,X_train,y_train,cv=5,scoring='accuracy')
#testing
knn.fit(X_train,y_train)
knnTest= knn.score(X_test,y_test)
```

Figure 4 below shows the different models tested, and their accuracy among the training and test sets. The GBM model performed the best at about 80% on both sets.

[31]:
```
#Makng table of training and testing accuracy of models tested
data = {
    'Model Name': ['GBM', 'Random Forest', 'TREE', 'KNN'],
    'Training Accuracy': [np.mean(gresults['test_score']), np.
 ↪mean(rf_results['test_score']), cv_acc, np.mean(kresults['test_score'])],
    'Testing Accuracy': [gbmtest, rfTest, val_acc, knnTest]
}

# Create a DataFrame
model_df = pd.DataFrame(data)

# Display the DataFrame
print(model_df)
```

```
      Model Name  Training Accuracy  Testing Accuracy
0            GBM           0.798406          0.802076
1  Random Forest           0.636511          0.634409
2           TREE           0.656862          0.672418
3            KNN           0.604376          0.627157
```

Figure 4 (Testing and Training Accuracy of Models Tested)

Figure 5 shows the confusion matrix for the selected GBM model. The model had a greater than 50% recall with every pitch, except the 2-seam fastball (only 34%). Looking at the plot, you can see the model predicted a 4-seam fastball 62% of the time, when it was actually a 2-seamer. Some other notable pitches that the model struggled differentiating with the 4-seamer were the cutter and sinker. Although it correctly predicted the cutter 58% of the time it also guessed 4-seam fastball (as well as slider) about 20% of the time. The sinker was also mistaken for a 4-seam fastball 39% of the time.

Since about 90% of MLB pitchers throw the 4-seamer, pitchers that can mix in an effective 2-seamer, sinker, and/or cutter potentially have an advanatge since these pitches seem like they can create extra confusion for hitters, when paired with the 4-seam fastball.

[42]:
```
from sklearn.metrics import plot_confusion_matrix
fig, axes = plt.subplots(1, 2, figsize=(20, 8))

# Plotting normalized confusion matrix
gbmmatrixpercentage = plot_confusion_matrix(gbm, X_test, y_test,␣
 ↪xticks_rotation='vertical', cmap='cividis', normalize='true', ax=axes[0])
gbmmatrixpercentage.ax_.set_title('Normalized Confusion Matrix')

# Plotting non-normalized confusion matrix
gbmmatrix = plot_confusion_matrix(gbm, X_test, y_test,␣
 ↪xticks_rotation='vertical', cmap='cividis', ax=axes[1])
gbmmatrix.ax_.set_title('Confusion Matrix')


# Adjust layout
plt.tight_layout()
plt.show()
```

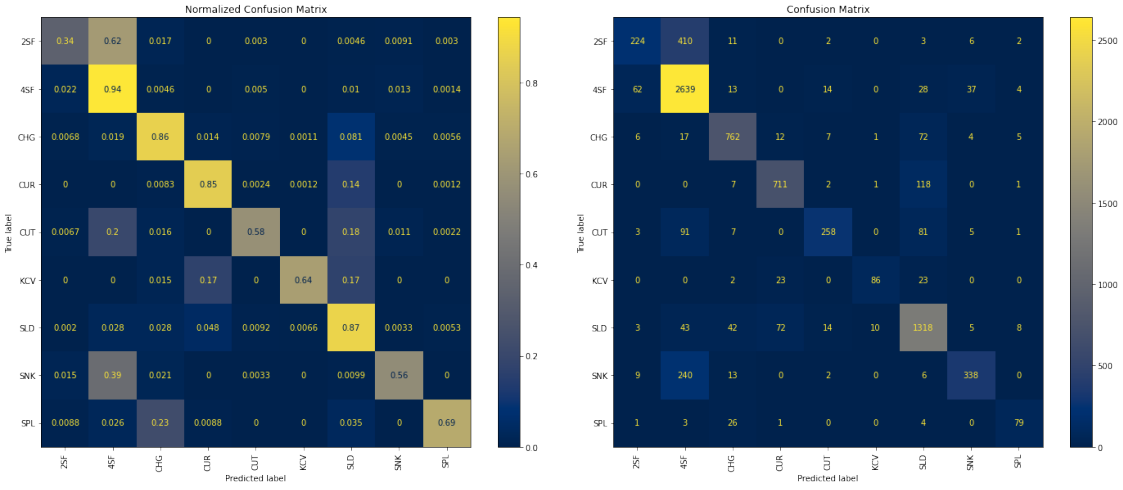


Figure 5 (Normalized and Non-Normalized Confusion Matrix for GBM Model)

Figure 6 shows the precision score for each pitch: Of all the times it was predicted, how often was that the true pitch? The precision scores for all pitches were greater than 70%. The only pitch with a signifigant decraese (compared to the recall on the conusion matrix) was the 4-seam fastball, since the model predcited 4-seamer on other pitches as well.

[43]:
```
from sklearn.metrics import precision_score
gbm_precision=pd.DataFrame([precision_score(y_test,gbm.
 ↪predict(X_test),average=None)],columns=sorted(y_test.unique()))
gbm_precision
```

[43]:

	2SF	4SF	CHG	CUR	CUT	KCV	SLD	\
0	0.727273	0.766483	0.862967	0.868132	0.862876	0.877551	0.797338	

	SNK	SPL
0	0.855696	0.79

Figure 6 (Precision Scores for GBM Model)

Figure 7 shows the variable importance for the selected GBM model. The most important variables were release speed and release spin rate, with other variables such as player name, and release position playing a role in predictions. One note here is that situational varaibles didn't play a huge role in predicting pitches.

[45]:
```
importances = gbm.feature_importances_

# Get feature names
feature_names = X_train.columns

# Sort importances in descending order
indices = np.argsort(importances)[::-1]

# Plot
plt.figure(figsize=(11, 8))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), feature_names[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
plt.show()
```

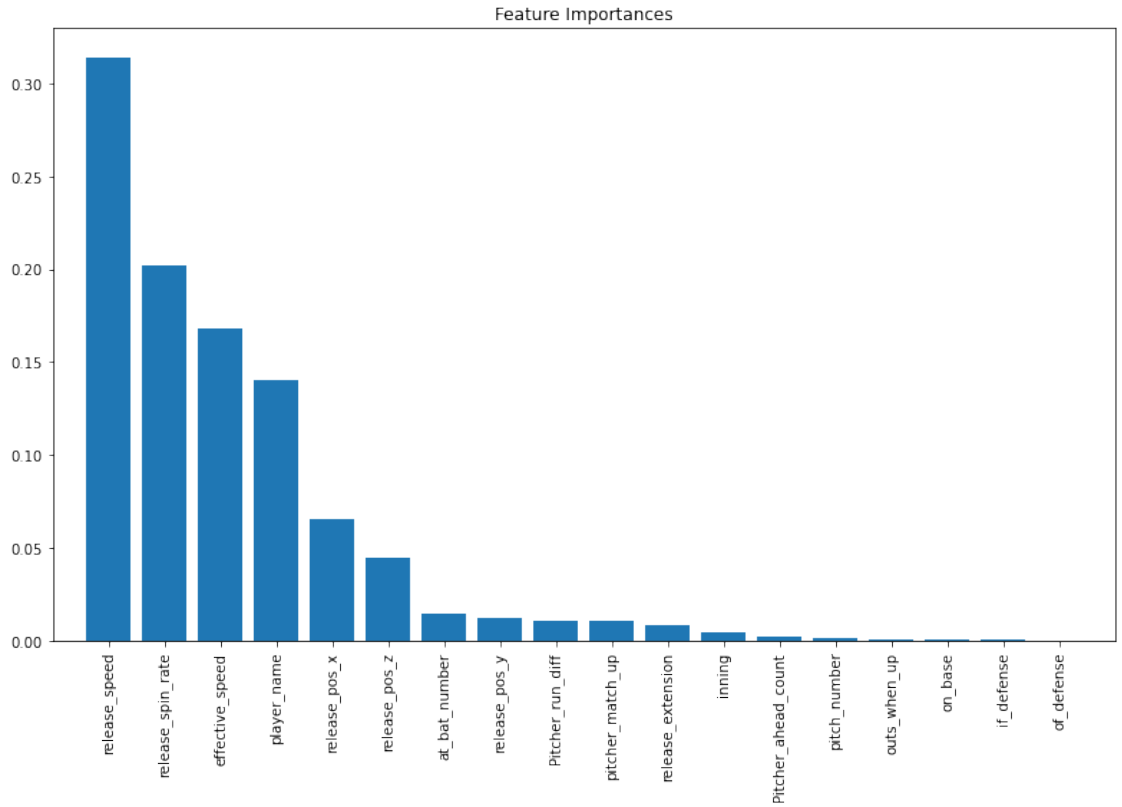


Figure 7 (Variable Importance for GBM Model)

Looking further into predicting pitches, knowing who is on the mound is crucial, since most pitchers have a select group of pitches that they throw, and have favorites as well. The number below indicates how many different pitches the average MLB player threw during this period.

[46]:
```
player_pitch_stats = df.groupby('player_name')['pitch_type'].nunique().
 ↪reset_index()

# Calculating the average number of unique pitch types
average_unique_pitch_types = player_pitch_stats['pitch_type'].mean()

print("The Average MLB pitcher has this many pitches in their arsenal:",␣
 ↪average_unique_pitch_types)
```

```
The Average MLB pitcher has this many pitches in their arsenal:
3.683333333333333
```

With the typical MLB player throwing 3-4 different pitch types, a hitter has to consider what pitches the pitcher likes to lean on, and how they can spot the difference between them using the important variables discussed. Two players were randomly selected to do further analysis to see how we can further predict pitches, given who is on the mound. As you can see in Figure 8, "Player 190" threw four different pitch types and threw offspeed a majorty of the time, but also mixed in

some fastballs as well. In Figure 9, you can see "Player 65" threw three different pitch types, with the 2-seamer being their favorite.

[47]:
```
#Dataframe of just one player, then grouping by pitch type
player_190 = df[df['player_name'] == 190]
pitch_stats = player_190.groupby('pitch_type').agg(
    pitch_count=('pitch_type', 'count'),
    avg_release_speed=('release_speed', 'mean'),
    avg_release_spin_rate=('release_spin_rate', 'mean')
)
pitch_stats_sorted = pitch_stats.sort_values(by='pitch_count', ascending=False)

print(pitch_stats_sorted)
```

```
            pitch_count  avg_release_speed  avg_release_spin_rate
pitch_type
CUR                  30          82.280000            2472.366667
CHG                  21          86.061905            1983.857143
2SF                  19          95.242105            2278.157895
4SF                  18          96.744444            2349.166667
```

Figure 8 ("Player 190" Pitches Sorted by Count and Release Speed and Spin Rate)

[48]:
```
player_65 = df[df['player_name'] == 65]
pitch_stats2 = player_65.groupby('pitch_type').agg(
    pitch_count=('pitch_type', 'count'),
    avg_release_speed=('release_speed', 'mean'),
    avg_release_spin_rate=('release_spin_rate', 'mean')
)
pitch_stats_sorted2 = pitch_stats2.sort_values(by='pitch_count',␣
↪ascending=False)

print(pitch_stats_sorted2)
```

```
            pitch_count  avg_release_speed  avg_release_spin_rate
pitch_type
2SF                  43          94.516279            1913.976744
SLD                  29          87.020690            2294.068966
SPL                  20          88.345000            1625.041505
```

Figure 9 ("Player 65" Pitches Sorted by Count and Release Speed and Spin Rate)

Another important variable in predicting pitches was release position. This is likely hard for the batter to pick up on in a split second, but Figure 10 below shows the different release points for the two selected players and how they vary for each pitch type. Player 1 tends to have a higher release on 2-seamers, while Player 2 has a lower release on their changeups.

[50]:
```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

```
# Grouping and assigning colors
color_list = list(mcolors.TABLEAU_COLORS.keys())
df3 =player_65.groupby("pitch_type").apply(lambda x: x.assign(color=color_list.
 ↪pop())).reset_index(drop=True)
df4 = player_190.groupby("pitch_type").apply(lambda x: x.
 ↪assign(color=color_list.pop())).reset_index(drop=True)

# Create a figure with two subplots side by side
fig, axes = plt.subplots(1, 2, figsize=(27, 14))

# Plot for the first player
axes[0].set_title("Player 65 - Pitch Type and Location")
for _, group in df3.groupby('pitch_type'):
    axes[0].scatter(group["release_pos_x"], group["release_pos_z"], s=50,␣
 ↪label=group['pitch_type'].iloc[0], color=group['color'])
axes[0].set_xlabel("Release Position X")
axes[0].set_ylabel("Release Position Z")
axes[0].legend(title='Pitch Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Plot for the second player
axes[1].set_title("Player 190 - Pitch Type and Location")
for _, group in df4.groupby('pitch_type'):
    axes[1].scatter(group["release_pos_x"], group["release_pos_z"], s=50,␣
 ↪label=group['pitch_type'].iloc[0], color=group['color'])
axes[1].set_xlabel("Release Position X")
axes[1].set_ylabel("Release Position Z")
axes[1].legend(title='Pitch Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Adjust layout
plt.tight_layout()
plt.show()
```

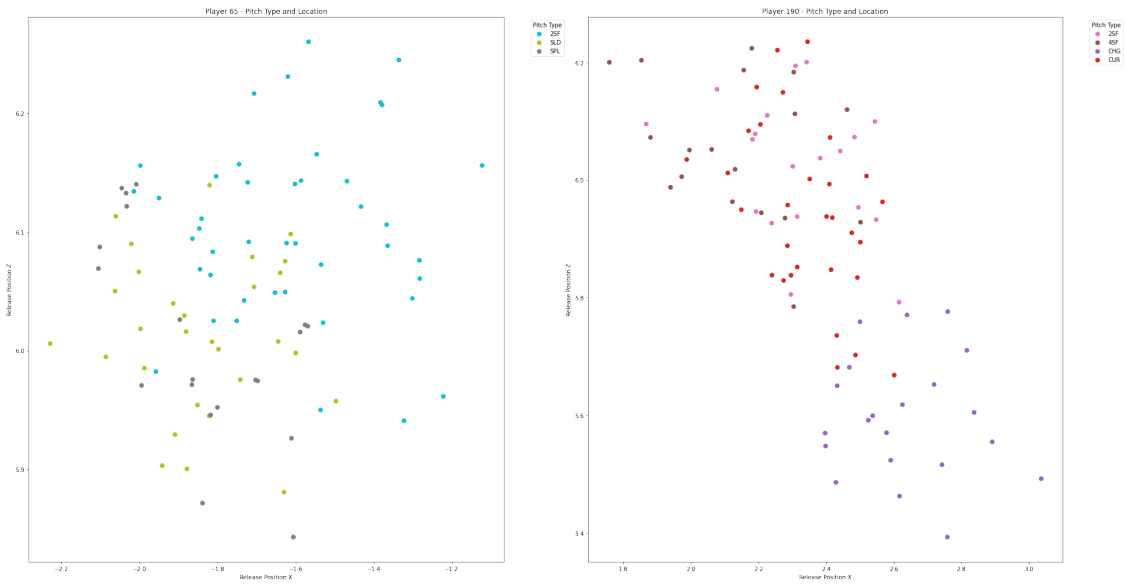


Figure 10 (Pitches for Two Selected Players, by Pitch Type and Release Position)