Predictive Modeling for Dodgers Big Three Batting Outcomes

In this notebook, I explore predictive models aimed at forecasting whether Mookie Betts, Shohei Ohtani, and Freddie Freeman will record a hit in MLB games. This analysis supports wagers on player performances, a feature offered by platforms such as FanDuel.

Introduction

FanDuel allows users to bet on various sports outcomes, including whether a specific MLB player will achieve a hit during a game. My personal experiences with these bets have yielded mixed results, which I attribute primarily to luck. Recognizing the inherent risks posed by the Law of Large Numbers, I am compelled to develop a more systematic and reliable predictive model to enhance the accuracy of my betting strategies over the long term.

Rationale For Player Selection

The focus on Mookie Betts, Shohei Ohtani, and Freddie Freeman is twofold. Firstly, as a Dodgers enthusiast, my betting interest naturally gravitates towards games featuring this team, making the choice to analyze these players both practical and enjoyable. Secondly, their secured long-term contracts with the Dodgers ensure a stable dataset for ongoing analysis. This stability is critical for developing a robust model that requires consistent performance data over multiple seasons.

By concentrating on these three prominent Dodgers players, I aim to create specialized models that can offer insights specific to their performance trends, thereby providing a strategic edge in sports betting focused on these athletes.

Data Collection Methodology

For the purpose of this project, the primary dataset will be sourced from pitch-level and game-level records, which provide comprehensive details relevant to predicting the outcomes of a player's plate appearance. This data is mcollected by Statcast, a high-resolution, automated tool developed to analyze player movements and game dynamics in Major League Baseball.

To facilitate efficient and effective data retrieval, I will utilize the pybaseball library, a widely recognized tool in the baseball analytics community. This library grants direct access to Statcast's database, allowing us to fetch detailed metrics such as pitch type, pitch velocity, player positions, hit outcomes, and other variables critical to our analysis.

In [ ]:
```
# Import the relavent libraries
from pybaseball import playerid_lookup, statcast_batter, statcast_pitcher, batting_stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objs as go
pyo.init_notebook_mode()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from imblearn.under_sampling import RandomUnderSampler
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA

%matplotlib inline
```

In [ ]:
```
# Look up Mookie Betts' player ID
player1 = playerid_lookup('betts', 'mookie')
mookie_id = player1['key_mlbam'].iloc[0]

# I decided to only use data as far back as 2018 and end at 2023
start_date = '2018-04-01'
end_date = '2023-9-30'

# Fetch pitch-level batting data
mookie_batting_data = statcast_batter(start_dt=start_date, end_dt=end_date, player_id=mo
okie_id)

print("Total elements in Mookie Betts' data frame:", mookie_batting_data.size)

print(mookie_batting_data.loc[0, ['player_name', 'home_team','away_team','inning_topbo
t']])
```

```
Gathering Player Data
Total elements in Mookie Betts' data frame: 1409154
player_name      Betts, Mookie
home_team                   SF
away_team                  LAD
inning_topbot              Top
Name: 0, dtype: object
```

In [ ]:
```
# Look up Shohei Ohtani's player ID
player2 = playerid_lookup('ohtani', 'shohei')
ohtani_id = player2['key_mlbam'].iloc[0]

start_date = '2018-04-01'
end_date = '2023-9-30'

# Fetch pitch-level batting data
ohtani_batting_data = statcast_batter(start_dt=start_date, end_dt=end_date, player_id=oh
tani_id)

print("Total elements in Shohei Ohtani's data frame:", ohtani_batting_data.size)

print(ohtani_batting_data.loc[0, ['player_name', 'home_team','away_team','inning_topbo
t']])
```

```
Gathering Player Data
Total elements in Shohei Ohtani's data frame: 1091998
player_name      Ohtani, Shohei
home_team                   OAK
away_team                   LAA
inning_topbot               Top
Name: 0, dtype: object
```

In [ ]:
```
# Look up Freddie Freeman's player ID
player3 = playerid_lookup('freeman', 'freddie')
freeman_id = player3['key_mlbam'].iloc[0]

start_date = '2018-04-01'
end_date = '2023-9-30'

# Fetch pitch-level batting data
freeman_batting_data = statcast_batter(start_dt=start_date, end_dt=end_date, player_id=f
reeman_id)

print("Total elements in Freddie Freeman's data frame:", freeman_batting_data.size)

print(freeman_batting_data.loc[0, ['player_name', 'home_team','away_team','inning_topbo
t']])
```

```
Gathering Player Data
Total elements in Freddie Freeman's data frame: 1474014
player_name      Freeman, Freddie
home_team                      SF
away_team                     LAD
inning_topbot                 Top
Name: 0, dtype: object
```

Preliminary Analysis

Upon initial examination of the data, it is evident that over the past five years, each of the three players—Mookie Betts, Shohei Ohtani, and Freddie Freeman—has encountered over a million pitches. Notably, Ohtani's pitch count is marginally lower in comparison to Betts and Freeman. This discrepancy can largely be attributed to Ohtani's more frequent injuries and his absence from playoff games, whereas Betts and Freeman have participated in multiple postseasons, thereby increasing their pitch encounters.

To verify the accuracy of the player data retrieved, I examined specific columns from the first row of the dataset. This step was crucial to ensure that the data pertained to the intended players and not to any other individuals with similar names.

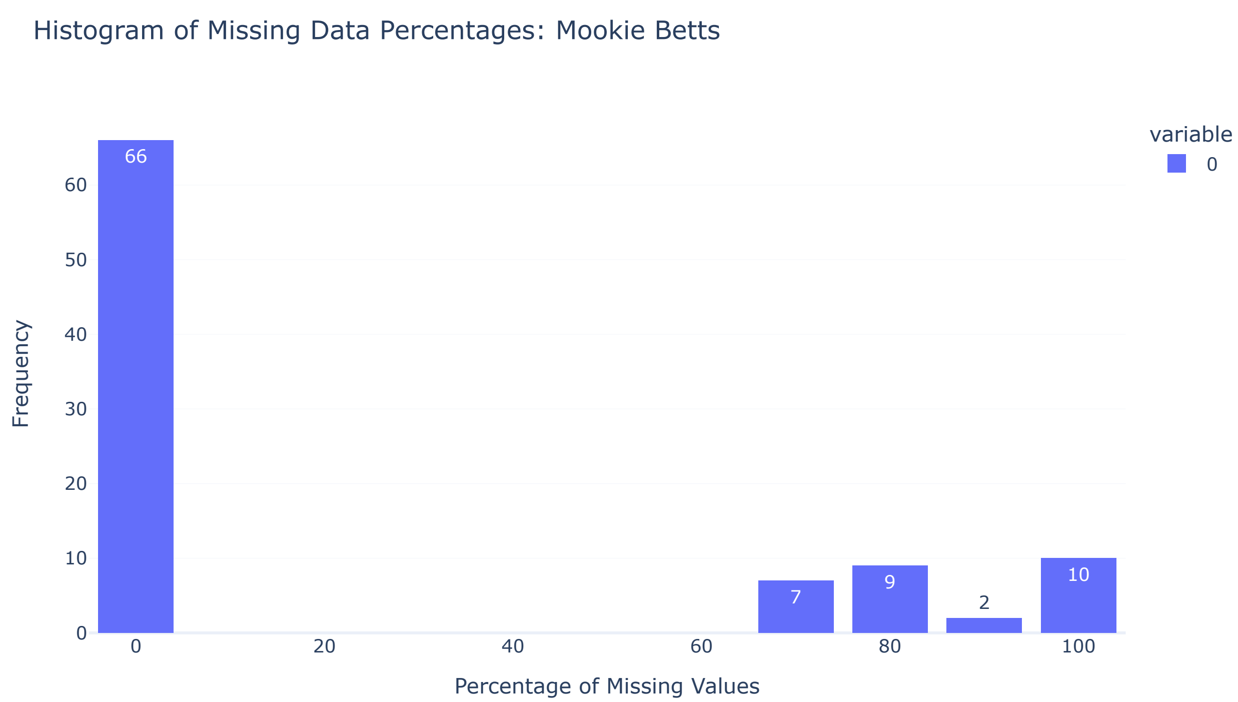
Further scrutiny of the dataset within the data viewer and consultation of the pybaseball documentation revealed the presence of 92 columns. These columns encompass a broad spectrum of game and pitch details, including information about the batter, pitcher, inning specifics, number of outs, base occupancy, pitch type, pitch location across the strike zone, outcome of the pitch, type of batted ball, among other variables. This dataset provides a robust foundation for developing predictive models that can accurately forecast player performance based on detailed game dynamics.

EDA

NA values

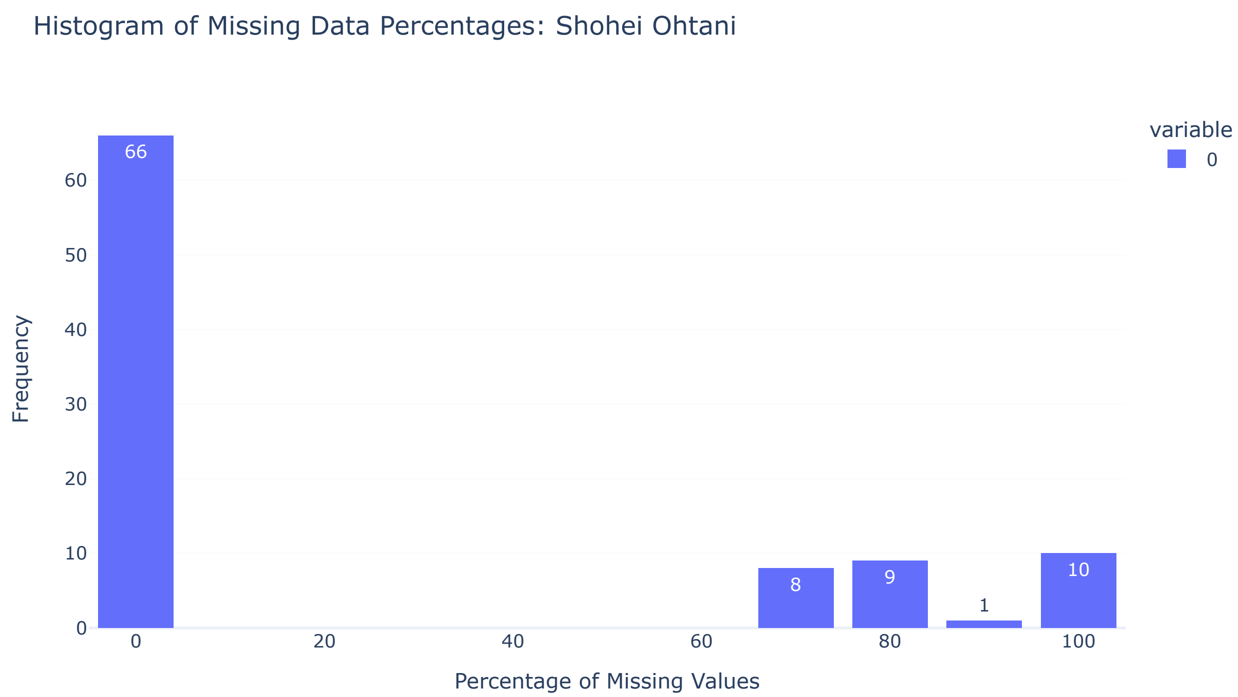
In [ ]:
```
mookie_data = pd.DataFrame(mookie_batting_data)

missing_percentage = mookie_data.isnull().sum() / len(mookie_data) * 100
missing_percentage_sorted = missing_percentage.sort_values(ascending=False)
print("Percentage of missing values per column:")
print(missing_percentage_sorted)

# Create a histogram of missingness in data set
fig = px.histogram(missing_percentage,
                   nbins=20,
                   title='Histogram of Missing Data Percentages: Mookie Betts',
                   labels={'value': 'Percentage of Missing Values'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Percentage of Missing Values",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

```
Percentage of missing values per column:
swing_length           100.0
bat_speed              100.0
tfs_deprecated         100.0
tfs_zulu_deprecated    100.0
umpire                 100.0
                        ...
inning                   0.0
fielder_8                0.0
fielder_9                0.0
home_team                0.0
fielder_7                0.0
Length: 94, dtype: float64
```



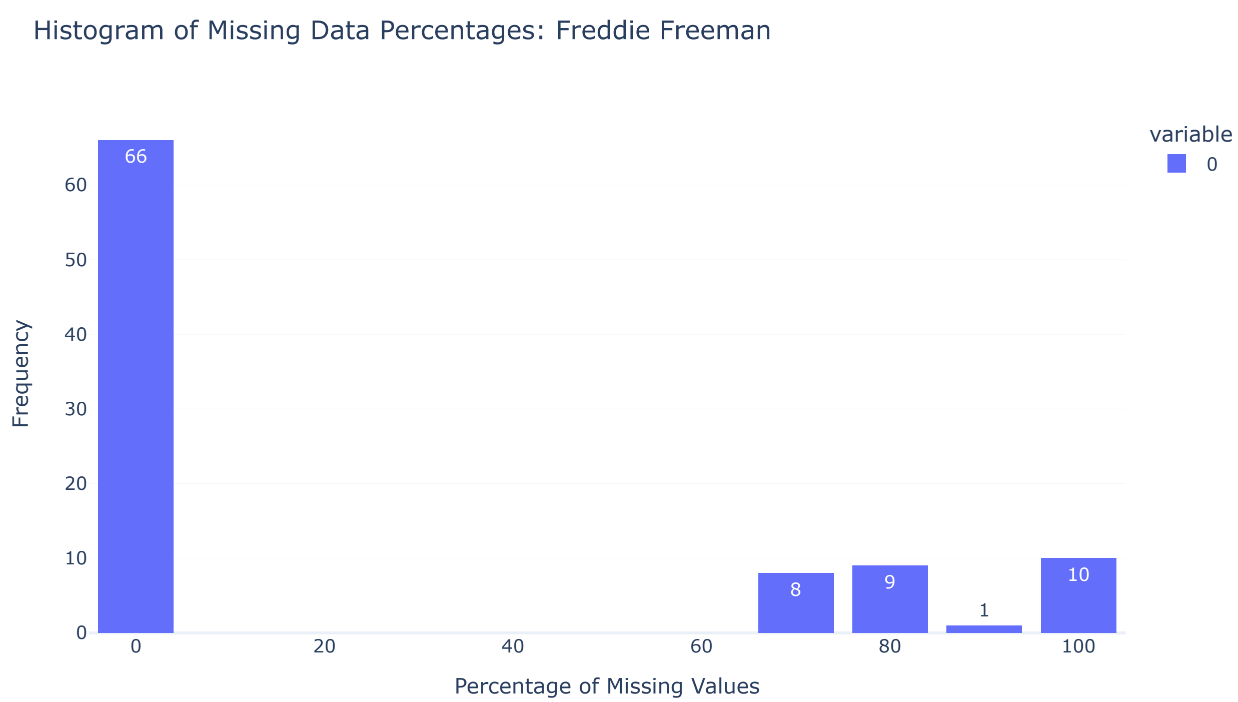
In [ ]:
```
shohei_data = pd.DataFrame(ohtani_batting_data)

missing_percentage1 = shohei_data.isnull().sum() / len(shohei_data) * 100
missing_percentage_sorted1 = missing_percentage1.sort_values(ascending=False)
print("Percentage of missing values per column:")
print(missing_percentage_sorted1)

# Create a histogram of missingness in data set
fig = px.histogram(missing_percentage1,
                   nbins=20,
                   title='Histogram of Missing Data Percentages: Shohei Ohtani',
                   labels={'value': 'Percentage of Missing Values'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Percentage of Missing Values",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

```
Percentage of missing values per column:
swing_length           100.0
bat_speed              100.0
tfs_zulu_deprecated    100.0
umpire                 100.0
sv_id                  100.0
                        ...
fielder_2                0.0
balls                    0.0
type                     0.0
away_team                0.0
fielder_9                0.0
Length: 94, dtype: float64
```



In [ ]:
```
freddie_data = pd.DataFrame(freeman_batting_data)

missing_percentage2 = freddie_data.isnull().sum() / len(freddie_data) * 100
missing_percentage_sorted2 = missing_percentage2.sort_values(ascending=False)
print("Percentage of missing values per column:")
print(missing_percentage_sorted2)

# Create a histogram of missingness in data set
fig = px.histogram(missing_percentage2,
                   nbins=20,
                   title='Histogram of Missing Data Percentages: Freddie Freeman',
                   labels={'value': 'Percentage of Missing Values'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Percentage of Missing Values",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

```
Percentage of missing values per column:
swing_length           100.0
spin_dir               100.0
umpire                 100.0
tfs_zulu_deprecated    100.0
tfs_deprecated         100.0
                       ...
fielder_8                0.0
fielder_9                0.0
away_team                0.0
inning_topbot            0.0
home_team                0.0
Length: 94, dtype: float64
```



Evaluation of Data Completeness

In the preliminary phase of our exploratory data analysis (EDA), we focused on assessing the extent of missing data within the dataset. The analysis began by generating histograms to visualize the distribution of missing data across various columns, revealing that the incidence of missing values does not vary significantly between different players. This uniformity suggests that missingness is likely influenced by factors independent of individual player characteristics.

Key Observations:

General Data Integrity

A promising finding is that 66 of the columns exhibited little (about 2%) or no missing values, indicating a high level of data completeness in these areas. The columns that contain around 2% missingness are mostly columns related to the properties of the pitch thrown such as release position, spin axis, and acceleration. Upon further inspection of the data, the missingness of those columns all have game dates that are in March and February which indicates that they are spring training games. This can be explained by knowing that not all fields where the MLB plays spring training are equipped with the sophisticated technology required to measure all of the details associated with a pitch. I will most likely not be including spring training data in my models so this missingness is a non-issue.

Complete Missingness in Legacy Fields

Eight columns showed 100% missing data. Further investigation revealed that these columns pertain to measurements from an outdated tracking system previously utilized by Major League Baseball (MLB). Modern equivalents of these measurements exist within the dataset, collected using updated technology, thereby ensuring that no critical information is missing despite the obsolescence of the older data columns.

Contextual Missingness

Columns exhibiting 70-90% missingness generally relate to events contingent on specific game situations, such as the batter making contact with the ball. Given that not all pitches result in contact, the high level of missingness in these columns is expected and logical. Similarly, columns recording the presence of runners on bases (first, second, or third) also showed substantial missingness, consistent with the frequent scenario of at-bats occurring with no runners on base.

Implications for Data Handling

The explanatory nature of missingness in this dataset obviates the need for imputation strategies such as mean or conditional mean filling. The absence of random missingness allows us to proceed with the available complete cases for most analyses, reducing the potential for bias that might arise from improper imputation.

Initial filtering

In [ ]:
```
# Drop deprecated and win expectancy columns

## Dropped win expectancy columns since they should have no predictive power
## in determing whether a player gets a hit.

mookie_data = mookie_data.drop(['break_length_deprecated', 'tfs_deprecated',
                                'tfs_zulu_deprecated','umpire','sv_id',
                                'spin_dir','spin_rate_deprecated','break_angle_deprecate
d',
                                'delta_run_exp', 'delta_home_win_exp'],
                                axis = 'columns')

shohei_data = shohei_data.drop(['break_length_deprecated', 'tfs_deprecated',
                                'tfs_zulu_deprecated','umpire','sv_id',
                                'spin_dir','spin_rate_deprecated','break_angle_deprecate
d',
                                'delta_run_exp', 'delta_home_win_exp'],
                                axis = 'columns')

freddie_data = freddie_data.drop(['break_length_deprecated', 'tfs_deprecated',
                                'tfs_zulu_deprecated','umpire','sv_id',
                                'spin_dir','spin_rate_deprecated','break_angle_deprecate
d',
                                'delta_run_exp', 'delta_home_win_exp'],
                                axis = 'columns')
```

In [ ]:
```
# Only look at regular season games

mookie_data = mookie_data.query("game_type == 'R'")

shohei_data = shohei_data.query("game_type == 'R'")

freddie_data = freddie_data.query("game_type == 'R'")
```

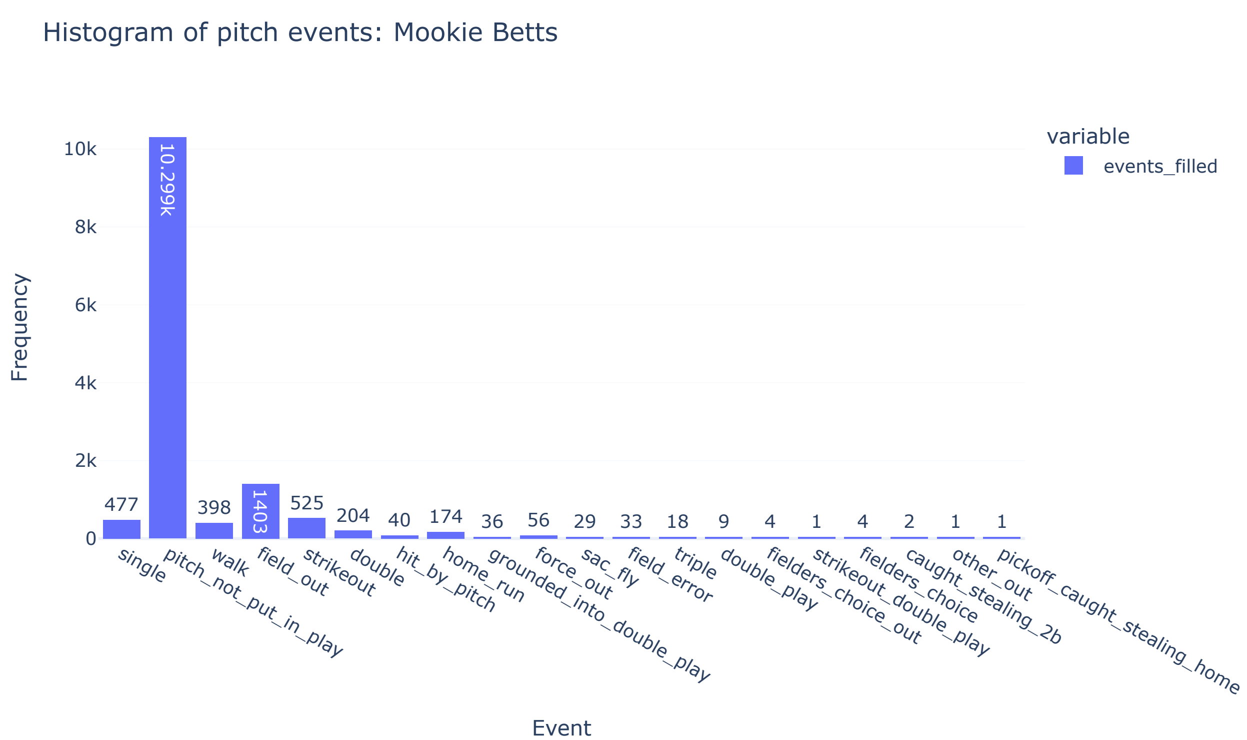
With the initial filtering out of the way I can now move on to more of the data visualization side of EDA

EDA Data Visualizations

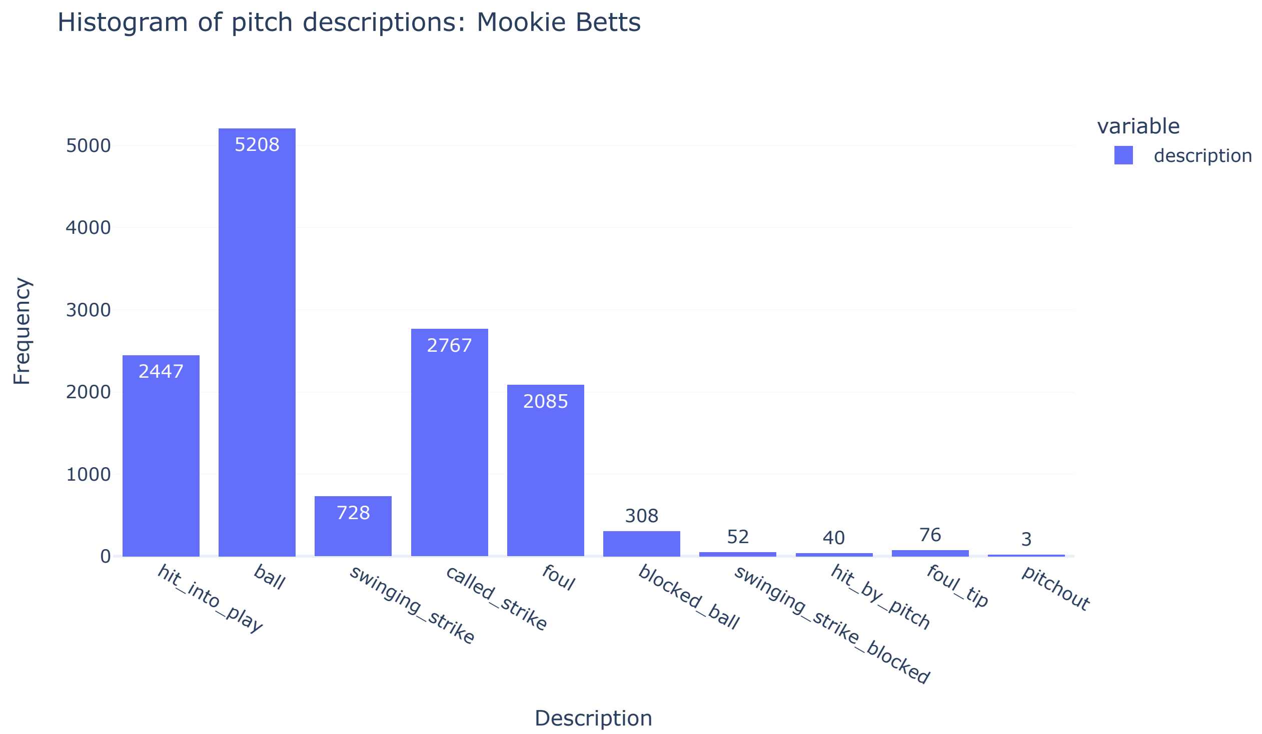
In [ ]:
```
# determine all of the unique events in the 'events' column
unique_events = mookie_data['events'].unique()
print(unique_events)

unique_description = mookie_data['description'].unique()
print(unique_description)
```

```
['single' nan 'walk' 'field_out' 'strikeout' 'double' 'hit_by_pitch'
 'home_run' 'grounded_into_double_play' 'force_out' 'sac_fly'
 'field_error' 'triple' 'double_play' 'fielders_choice_out'
 'strikeout_double_play' 'fielders_choice' 'caught_stealing_2b'
 'other_out' 'pickoff_caught_stealing_home']
['hit_into_play' 'ball' 'swinging_strike' 'called_strike' 'foul'
 'blocked_ball' 'swinging_strike_blocked' 'hit_by_pitch' 'foul_tip'
 'pitchout']
```

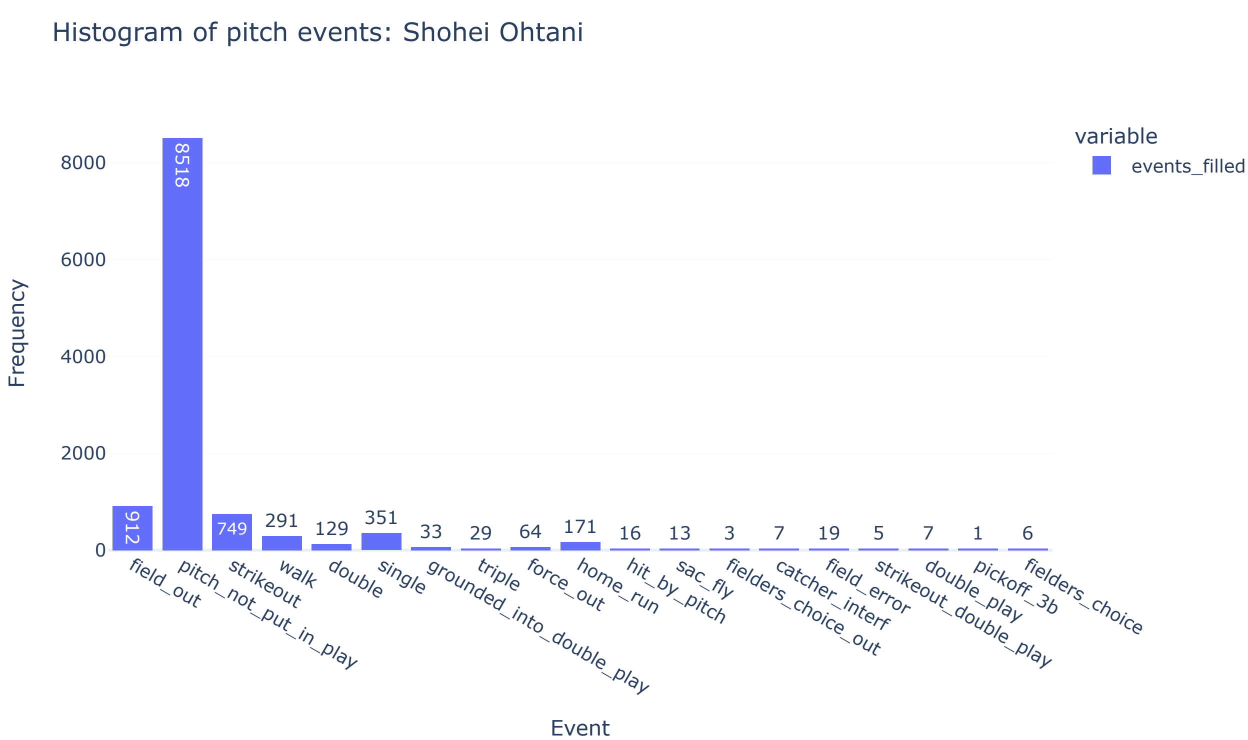
In [ ]:
```
mookie_data['events_filled'] = mookie_data['events'].fillna('pitch_not_put_in_play')

# create histogram of the events of every pitch
fig = px.histogram(mookie_data['events_filled'],
                   nbins=20,
                   title='Histogram of pitch events: Mookie Betts',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Event",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



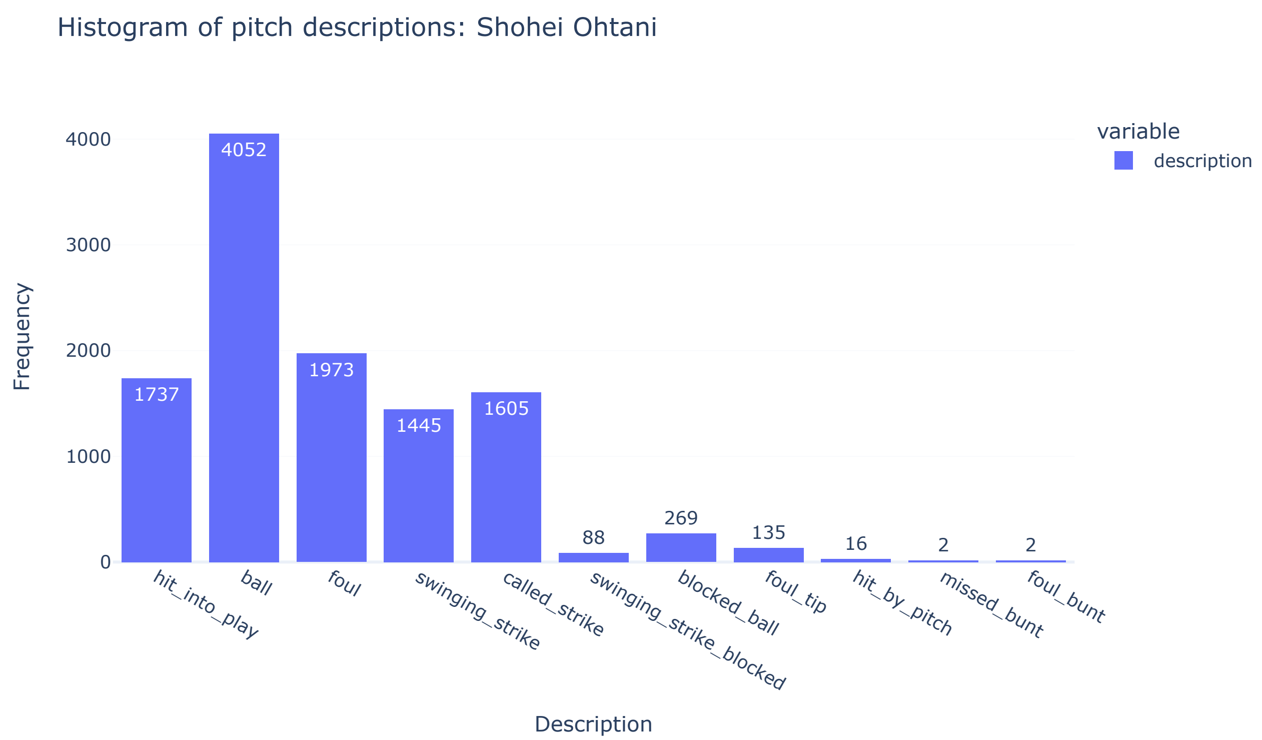
In [ ]:
```
# create histogram of the description of each pitch
fig = px.histogram(mookie_data['description'],
                   nbins=20,
                   title='Histogram of pitch descriptions: Mookie Betts',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Description",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



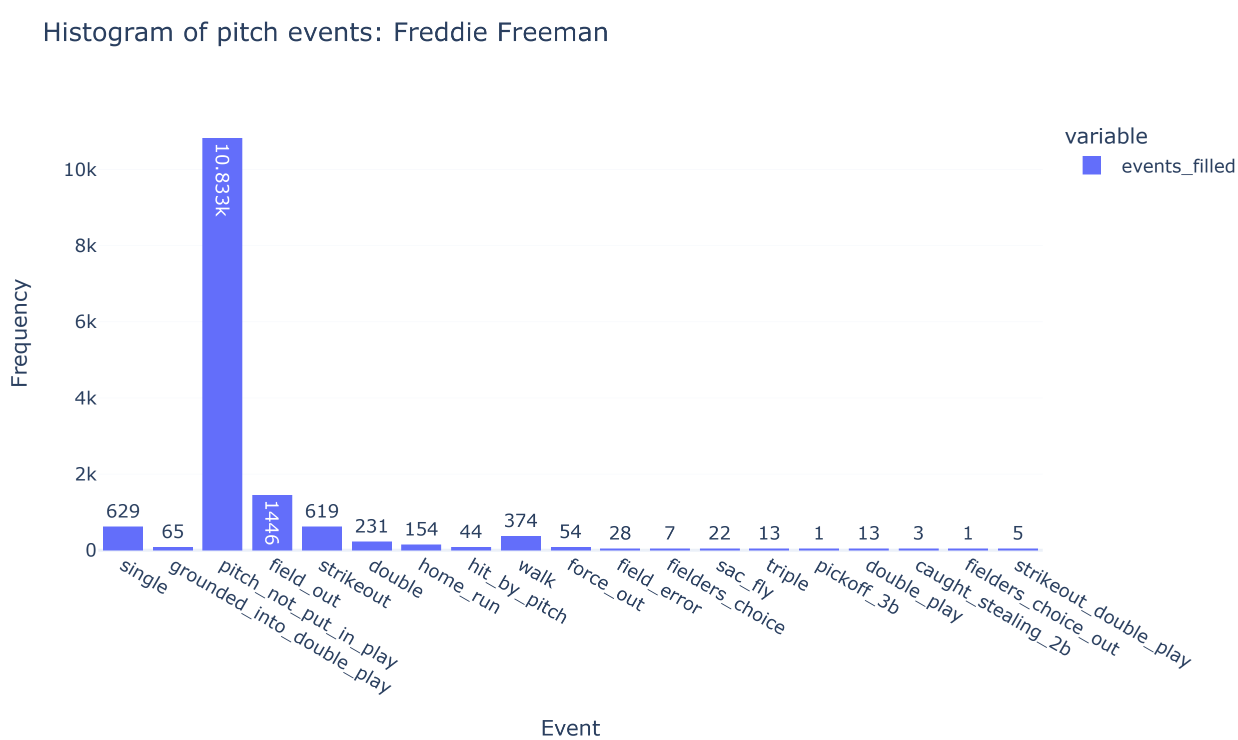
In [ ]:
```
shohei_data['events_filled'] = shohei_data['events'].fillna('pitch_not_put_in_play')

# create histogram of the events of every pitch
fig = px.histogram(shohei_data['events_filled'],
                   nbins=20,
                   title='Histogram of pitch events: Shohei Ohtani',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Event",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



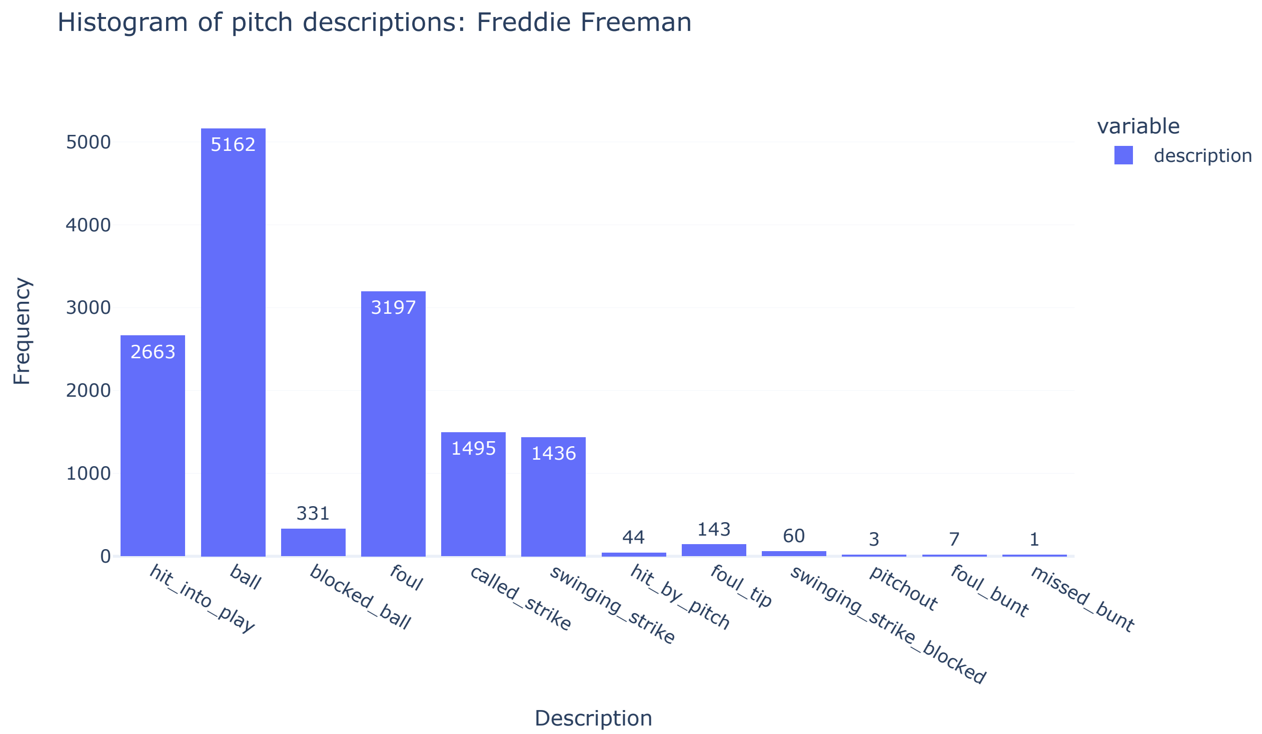
In [ ]:
```
# create histogram of the description of each pitch
fig = px.histogram(shohei_data['description'],
                   nbins=20,
                   title='Histogram of pitch descriptions: Shohei Ohtani',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Description",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



In [ ]:
```
freddie_data['events_filled'] = freddie_data['events'].fillna('pitch_not_put_in_play')

# create histogram of the events of every pitch
fig = px.histogram(freddie_data['events_filled'],
                   nbins=20,
                   title='Histogram of pitch events: Freddie Freeman',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Event",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



In [ ]:
```
# create histogram of the description of each pitch
fig = px.histogram(freddie_data['description'],
                   nbins=20,
                   title='Histogram of pitch descriptions: Freddie Freeman',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Description",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

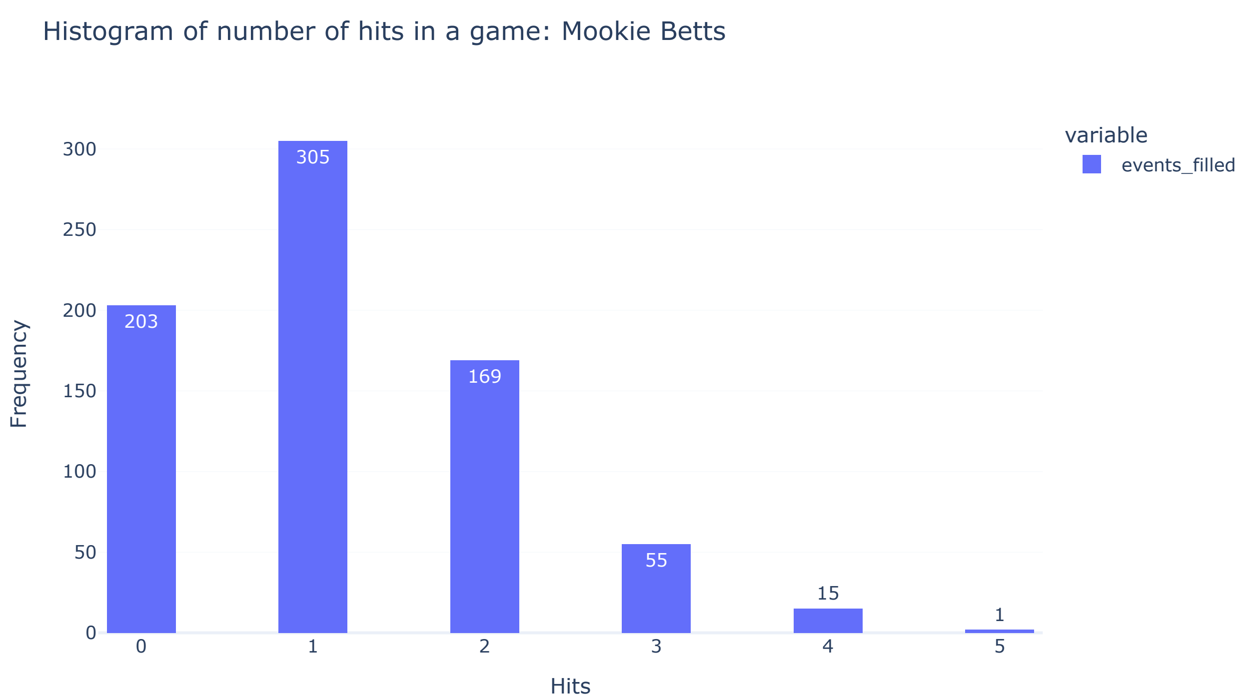


From the analysis of histograms representing pitch events and descriptions, the data appear to be consistent with expected baseball outcomes. The majority of pitches observed were not put into play, and the distributions of singles, doubles, triples, and home runs align with the expected performance of All-Star level players over a five-year period. Furthermore, the frequencies of balls, strikes, and balls put into play are also within anticipated ranges. While the exact figures vary among the players, the overall relative trends are consistent across the dataset.

With a comprehensive overview of the total counts of pitch results established, the next phase of the analysis involves examining these events on a per-game basis to gain insights into the rate of occurrences. This will allow for a more nuanced understanding of player performance dynamics and the efficiency of their play in games.

In [ ]:
```
mookie_filtered = mookie_data.dropna(subset = 'events')

def count_hits(outcome_list):
    hits = ['single', 'double', 'triple', 'home_run']
    count = 0
    for outcome in outcome_list:
        if outcome in hits:
            count += 1
    return count

mookie_games = mookie_filtered.groupby('game_pk', as_index=False).agg(count_hits)
```

In [ ]:
```
# create histogram of number of hits in a game
fig = px.histogram(mookie_games['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game: Mookie Betts',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

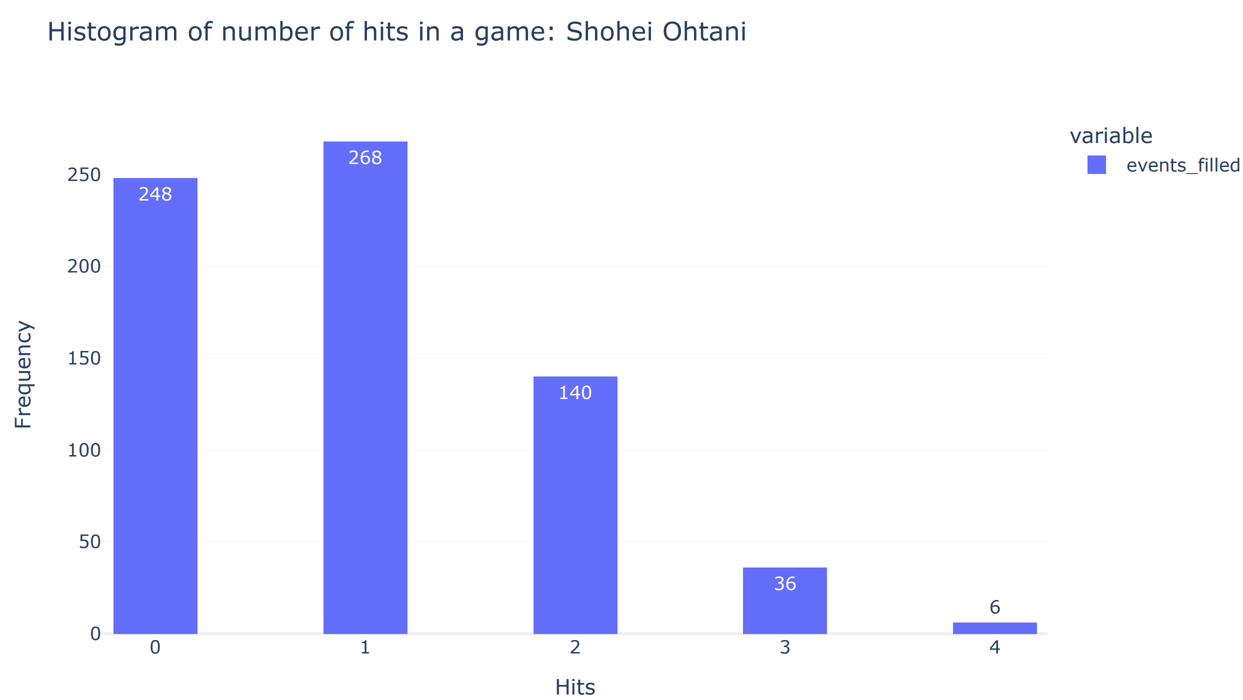


In [ ]:
```
mookie_value_counts = mookie_games['events_filled'].value_counts()

print(mookie_value_counts)

percent_no_hits = (mookie_value_counts[0] / mookie_value_counts.sum() * 100).round(2)

print("Percent of games with no hits: {}%".format(percent_no_hits))
```

```
1    305
0    203
2    169
3     55
4     15
5      1
Name: events_filled, dtype: int64
Percent of games with no hits: 27.14%
```

In [ ]:
```
shohei_filtered = shohei_data.dropna(subset = 'events')

shohei_games = shohei_filtered.groupby('game_pk', as_index=False).agg(count_hits)
```

In [ ]:
```
# create histogram of number of hits in a game
fig = px.histogram(shohei_games['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game: Shohei Ohtani',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```

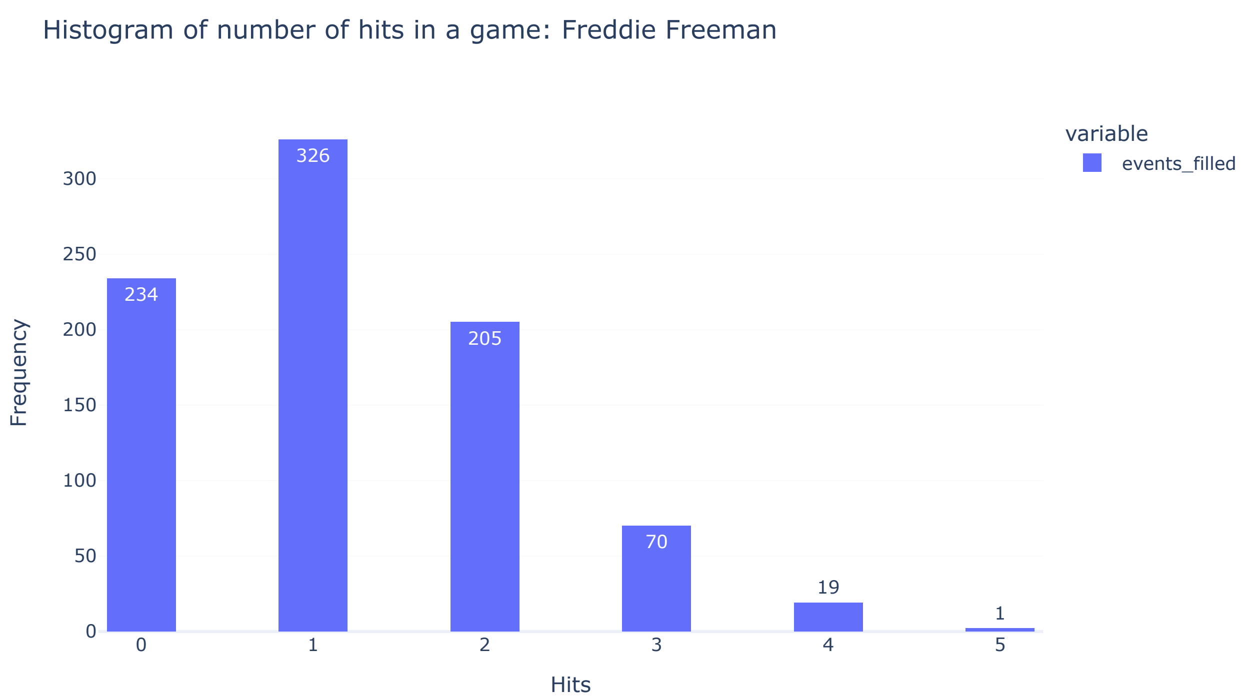


In [ ]:
```
shohei_value_counts = shohei_games['events_filled'].value_counts()

print(shohei_value_counts)

percent_no_hits = (shohei_value_counts[0] / shohei_value_counts.sum() * 100).round(2)

print("Percent of games with no hits: {}%".format(percent_no_hits))
```

```
1    268
0    248
2    140
3     36
4      6
Name: events_filled, dtype: int64
Percent of games with no hits: 35.53%
```

In [ ]:
```
freddie_filtered = freddie_data.dropna(subset = 'events')

freddie_games = freddie_filtered.groupby('game_pk', as_index=False).agg(count_hits)
```

In [ ]:
```
# create histogram of number of hits in a game
fig = px.histogram(freddie_games['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game: Freddie Freeman',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



In [ ]:
```
freddie_value_counts = freddie_games['events_filled'].value_counts()

print(freddie_value_counts)

percent_no_hits = (freddie_value_counts[0] / freddie_value_counts.sum() * 100).round(2)

print("Percent of games with no hits: {}%".format(percent_no_hits))
```

```
1    326
0    234
2    205
3     70
4     19
5      1
Name: events_filled, dtype: int64
Percent of games with no hits: 27.37%
```

Based on an initial analysis of game-level hits data for three baseball players, it appears that each player is more likely to record at least one hit in a game than not. Specifically, Betts and Freeman typically have games without hits approximately 27% of the time, while Ohtani records no hits in about 35% of games. These probabilities align closely with the current betting odds offered by FanDuel for these players to record a hit. Throughout this season, FanDuel has generally set the odds for all three players between -250 and -280, which implies a probability of 71.43-73.68% that the players will get a hit, or conversely, a 26.32-28.57% chance they will not.

In [ ]:
```
# determine all the unique pitches in the data set
unique_pitches = mookie_data['pitch_name'].unique()

print(unique_pitches)
```

```
['Sweeper' '4-Seam Fastball' 'Slider' 'Sinker' 'Split-Finger'
 'Knuckle Curve' 'Cutter' 'Changeup' 'Curveball' 'Slurve' 'Forkball'
 'Pitch Out' 'Screwball' 'Other' 'Slow Curve' 'Knuckleball' nan]
```

In [ ]:
```
# filter data according to pitch

# Mookie
mookie_fastball_filtered_data = mookie_data[mookie_data['pitch_name'] == '4-Seam Fastbal
l']

mookie_slider_filtered_data = mookie_data[mookie_data['pitch_name'] == 'Slider']

mookie_curveball_filtered_data = mookie_data[mookie_data['pitch_name'] == 'Curveball']

# Shohei
shohei_fastball_filtered_data = shohei_data[shohei_data['pitch_name'] == '4-Seam Fastbal
l']

shohei_slider_filtered_data = shohei_data[shohei_data['pitch_name'] == 'Slider']

shohei_curveball_filtered_data = shohei_data[shohei_data['pitch_name'] == 'Curveball']

# Freddie
freddie_fastball_filtered_data = freddie_data[freddie_data['pitch_name'] == '4-Seam Fast
ball']

freddie_slider_filtered_data = freddie_data[freddie_data['pitch_name'] == 'Slider']

freddie_curveball_filtered_data = freddie_data[freddie_data['pitch_name'] == 'Curvebal
l']
```

In [ ]:
```
hits = ['single', 'double', 'triple', 'home_run']

mookie_total_hits_fastball = mookie_fastball_filtered_data['events_filled'].isin(hits).s
um()

mookie_total_pitches_fastball = len(mookie_fastball_filtered_data)

print(f"Mookie Betts' (hits \ total pitches seen) against fastballs: {mookie_total_hits_
fastball/mookie_total_pitches_fastball:.3f}")

mookie_total_hits_slider = mookie_slider_filtered_data['events_filled'].isin(hits).sum()

mookie_total_pitches_slider = len(mookie_slider_filtered_data)

print(f"Mookie Betts' (hits \ total pitches seen) against sliders: {mookie_total_hits_sl
ider/mookie_total_pitches_slider:.3f}")

mookie_total_hits_curveball = mookie_curveball_filtered_data['events_filled'].isin(hit
s).sum()

mookie_total_pitches_curveball = len(mookie_curveball_filtered_data)

print(f"Mookie Betts' (hits \ total pitches seen) against curveballs: {mookie_total_hits
_curveball/mookie_total_pitches_curveball:.3f}")
```

```
Mookie Betts' (hits \ total pitches seen) against fastballs: 0.058
Mookie Betts' (hits \ total pitches seen) against sliders: 0.057
Mookie Betts' (hits \ total pitches seen) against curveballs: 0.049
```

In [ ]:
```
shohei_total_hits_fastball = shohei_fastball_filtered_data['events_filled'].isin(hits).sum()

shohei_total_pitches_fastball = len(shohei_fastball_filtered_data)

print(f"Shohei Ohtani's (hits \ total pitches seen) against fastballs: {shohei_total_hits_fastball/shohei_total_pitches_fastball:.3f}")

shohei_total_hits_slider = shohei_slider_filtered_data['events_filled'].isin(hits).sum()

shohei_total_pitches_slider = len(shohei_slider_filtered_data)

print(f"Shohei Ohtani's (hits \ total pitches seen) against sliders: {shohei_total_hits_slider/shohei_total_pitches_slider:.3f}")

shohei_total_hits_curveball = shohei_curveball_filtered_data['events_filled'].isin(hits).sum()

shohei_total_pitches_curveball = len(shohei_curveball_filtered_data)

print(f"Shohei Ohtani's (hits \ total pitches seen) against curveballs: {shohei_total_hits_curveball/shohei_total_pitches_curveball:.3f}")
```

```
Shohei Ohtani's (hits \ total pitches seen) against fastballs: 0.060
Shohei Ohtani's (hits \ total pitches seen) against sliders: 0.054
Shohei Ohtani's (hits \ total pitches seen) against curveballs: 0.055
```

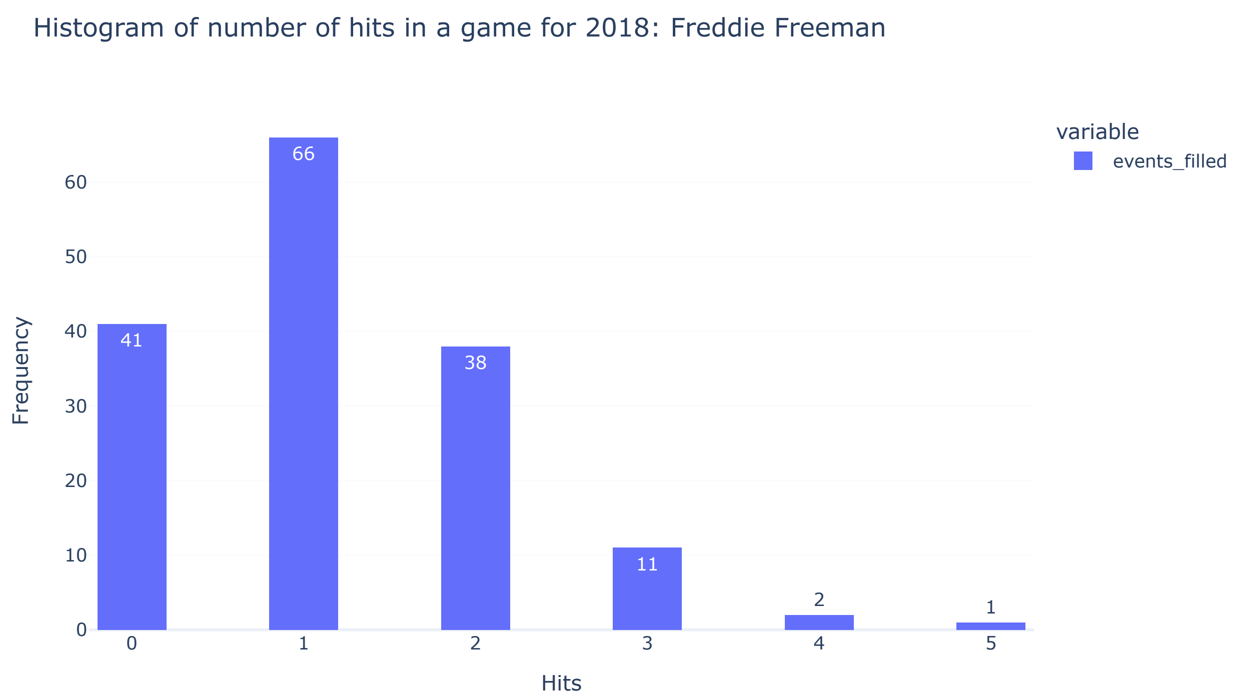
In [ ]:
```
freddie_total_hits_fastball = freddie_fastball_filtered_data['events_filled'].isin(hits).sum()

freddie_total_pitches_fastball = len(freddie_fastball_filtered_data)

print(f"Freddie Freeman's (hits \ total pitches seen) against fastballs: {freddie_total_hits_fastball/freddie_total_pitches_fastball:.3f}")

freddie_total_hits_slider = freddie_slider_filtered_data['events_filled'].isin(hits).sum()

freddie_total_pitches_slider = len(freddie_slider_filtered_data)

print(f"Freddie Freeman's (hits \ total pitches seen) against sliders: {freddie_total_hits_slider/freddie_total_pitches_slider:.3f}")

freddie_total_hits_curveball = freddie_curveball_filtered_data['events_filled'].isin(hits).sum()

freddie_total_pitches_curveball = len(freddie_curveball_filtered_data)

print(f"Freddie Freeman's (hits \ total pitches seen) against curveballs: {freddie_total_hits_curveball/freddie_total_pitches_curveball:.3f}")
```

```
Freddie Freeman's (hits \ total pitches seen) against fastballs: 0.079
Freddie Freeman's (hits \ total pitches seen) against sliders: 0.055
Freddie Freeman's (hits \ total pitches seen) against curveballs: 0.065
```

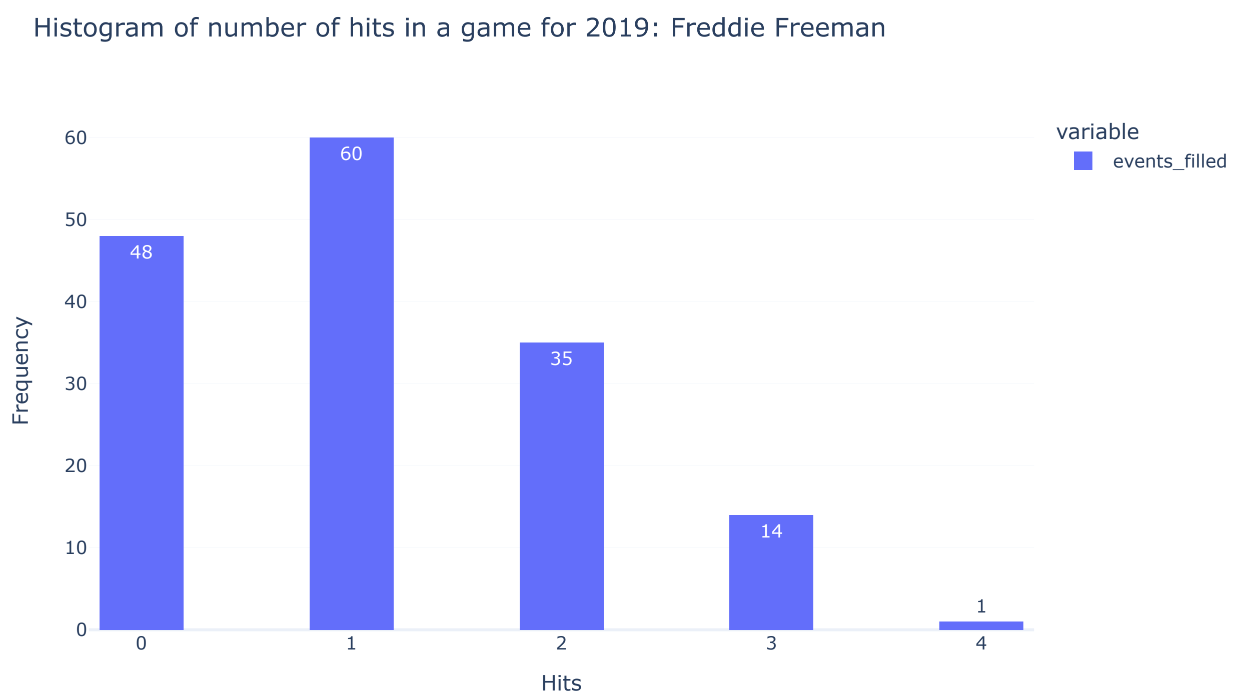
In the previous analysis, the ratio of hits to total pitches observed was calculated for three of the most common pitches faced by MLB players. The findings indicated that both Ohtani and Betts demonstrated hit rates of approximately 5-6% across the three pitch types, while Freddie Freeman exhibited a slightly higher range of 5.5-8%. This data is relavant for the development of predictive models that estimate the probability of getting a hit or expected number of hits using expected number of pitches and the types of pitches a batter is likely to encounter in their upcoming games.

In [ ]:
```
# create separate data set for each year
freddie_2018 = freddie_filtered[freddie_filtered['game_year'] == 2018]

freddie_2018 = freddie_2018.groupby('game_pk', as_index=False).agg(count_hits)

freddie_2019 = freddie_filtered[freddie_filtered['game_year'] == 2019]

freddie_2019 = freddie_2019.groupby('game_pk', as_index=False).agg(count_hits)

freddie_2020 = freddie_filtered[freddie_filtered['game_year'] == 2020]

freddie_2020 = freddie_2020.groupby('game_pk', as_index=False).agg(count_hits)

freddie_2021 = freddie_filtered[freddie_filtered['game_year'] == 2021]

freddie_2021 = freddie_2021.groupby('game_pk', as_index=False).agg(count_hits)

freddie_2022 = freddie_filtered[freddie_filtered['game_year'] == 2022]

freddie_2022 = freddie_2022.groupby('game_pk', as_index=False).agg(count_hits)

freddie_2023 = freddie_filtered[freddie_filtered['game_year'] == 2023]

freddie_2023 = freddie_2023.groupby('game_pk', as_index=False).agg(count_hits)
```

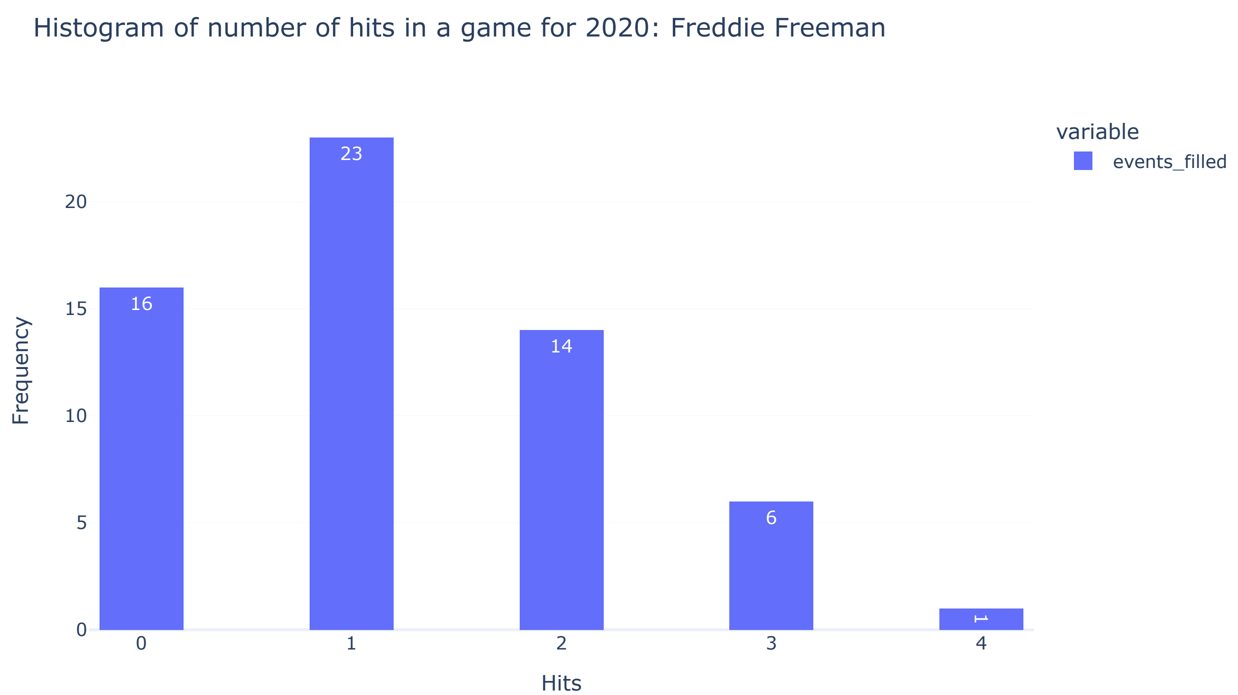
In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2018['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2018: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



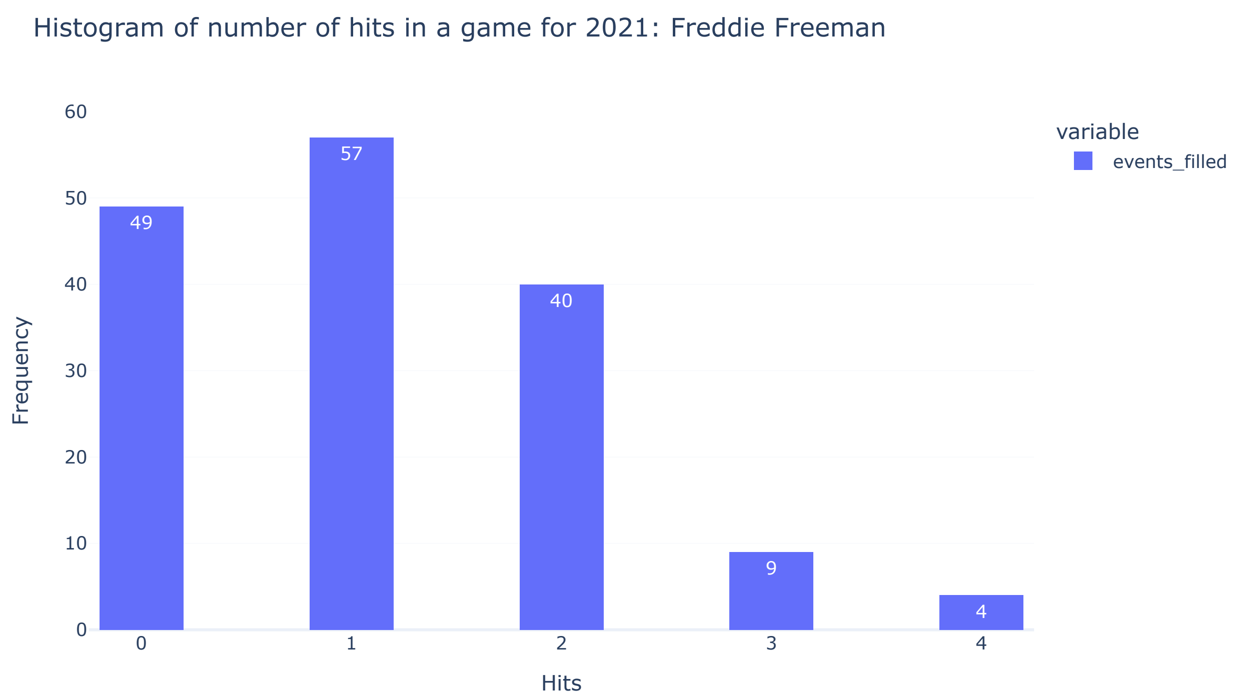
In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2019['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2019: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



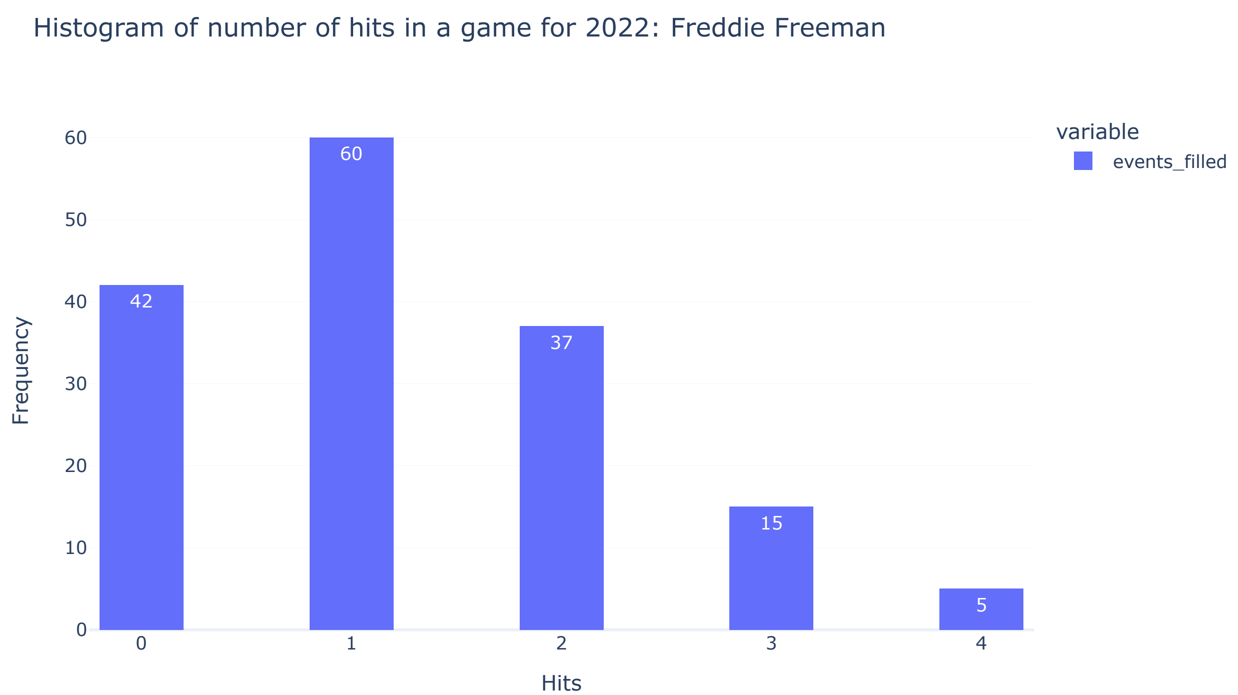
In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2020['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2020: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



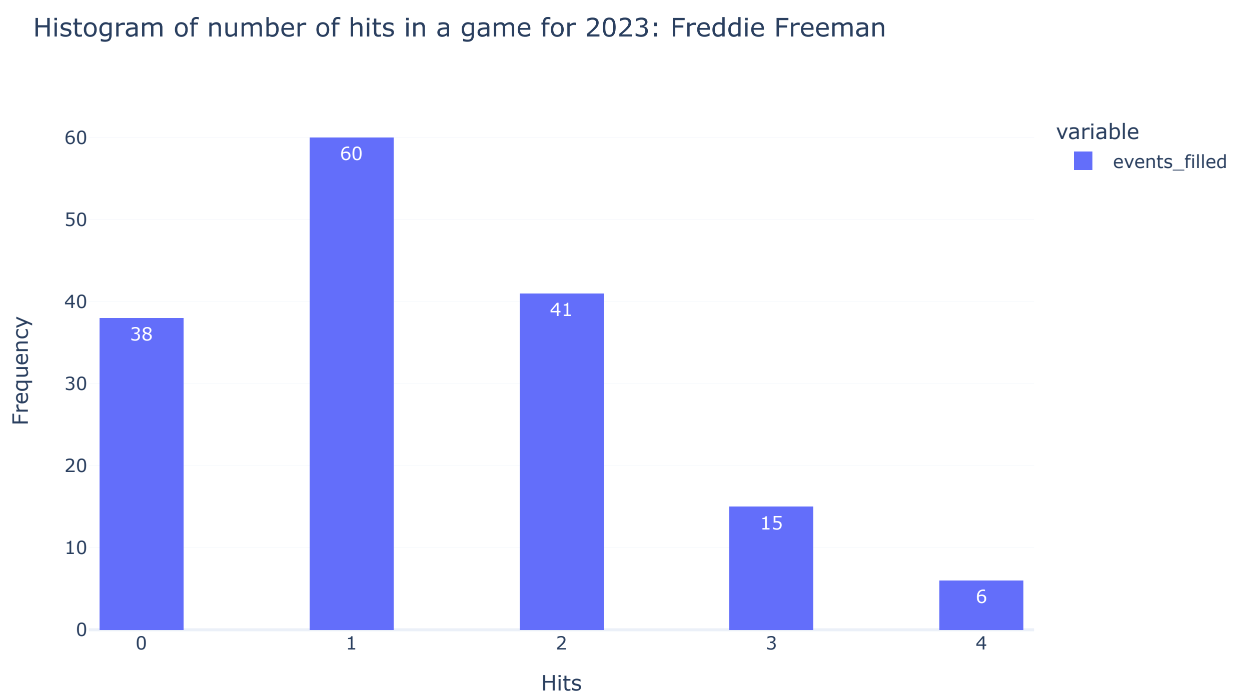
In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2021['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2021: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2022['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2022: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



In [ ]:
```
# create histogram of hits in given year
fig = px.histogram(freddie_2023['events_filled'],
                   nbins=20,
                   title='Histogram of number of hits in a game for 2023: Freddie Freema
n',
                   labels={'value': 'Frequency'},
                   text_auto=True,
                   template='plotly_white')

fig.update_layout(
    xaxis_title="Hits",
    yaxis_title="Frequency",
    bargap=0.2,
)

fig.write_html('plot.html')

fig.show()
```



The histograms depicting the distribution of hits per game for Freddie Freeman across different years were generated and analyzed. As anticipated, the results exhibited minimal variation between years, corroborating the hypothesis that batting performance remains relatively consistent annually. This finding supports the use of a simplifying assumption in statistical modeling that batting performance does not vary significantly by year. Subsequent verification was conducted using player statistics from Baseball Reference for Shohei Ohtani and Mookie Betts, further affirming that their performance metrics also show little annual variation. This consistency across multiple players strengthens the validity of the assumption for use in predictive modeling endeavors.

Models

In [ ]:
```
unique_events = mookie_data['events_filled'].unique()
print(unique_events)
```

```
['single' 'pitch_not_put_in_play' 'walk' 'field_out' 'strikeout' 'double'
 'hit_by_pitch' 'home_run' 'grounded_into_double_play' 'force_out'
 'sac_fly' 'field_error' 'triple' 'double_play' 'fielders_choice_out'
 'strikeout_double_play' 'fielders_choice' 'caught_stealing_2b'
 'other_out' 'pickoff_caught_stealing_home']
```

In [ ]:
```
hits = ['single', 'double', 'triple', 'home_run']

# create new column for classification of hit(s) or no hits game
mookie_data['is_hit'] = mookie_data['events_filled'].apply(lambda x: 1 if x in hits else
0)

shohei_data['is_hit'] = shohei_data['events_filled'].apply(lambda x: 1 if x in hits else
0)

freddie_data['is_hit'] = freddie_data['events_filled'].apply(lambda x: 1 if x in hits el
se 0)
```

In [ ]:
```
def categorize_count(row):
    if row['balls'] == 0 and row['strikes'] == 0:
        return '0-0'
    elif row['balls'] == 1 and row['strikes'] == 0:
        return '1-0'
    elif row['balls'] == 2 and row['strikes'] == 0:
        return '2-0'
    elif row['balls'] == 3 and row['strikes'] == 0:
        return '3-0'
    elif row['balls'] == 0 and row['strikes'] == 1:
        return '0-1'
    elif row['balls'] == 1 and row['strikes'] == 1:
        return '1-1'
    elif row['balls'] == 2 and row['strikes'] == 1:
        return '2-1'
    elif row['balls'] == 3 and row['strikes'] == 1:
        return '3-1'
    elif row['balls'] == 0 and row['strikes'] == 2:
        return '0-2'
    elif row['balls'] == 1 and row['strikes'] == 2:
        return '1-2'
    elif row['balls'] == 2 and row['strikes'] == 2:
        return '2-2'
    elif row['balls'] == 3 and row['strikes'] == 2:
        return '3-2'
    else:
        return 'Cooked'
```

In [ ]:
```
# create new column that represents the count right before pitch
mookie_data['count'] = mookie_data.apply(categorize_count, axis=1)

# only look at variables I believe have most predictive power on is_hit
mookie_data_relavent = mookie_data[['is_hit','count','pitch_type','p_throws','plate_
x','plate_z']]

# create dummies
categorical_vars = ['pitch_type', 'p_throws']

mookie_data_relavent = pd.get_dummies(mookie_data_relavent, columns=categorical_vars, dr
op_first=True)

mookie_data_relavent = pd.get_dummies(mookie_data_relavent, columns=['count'])
```

Logistic Regression

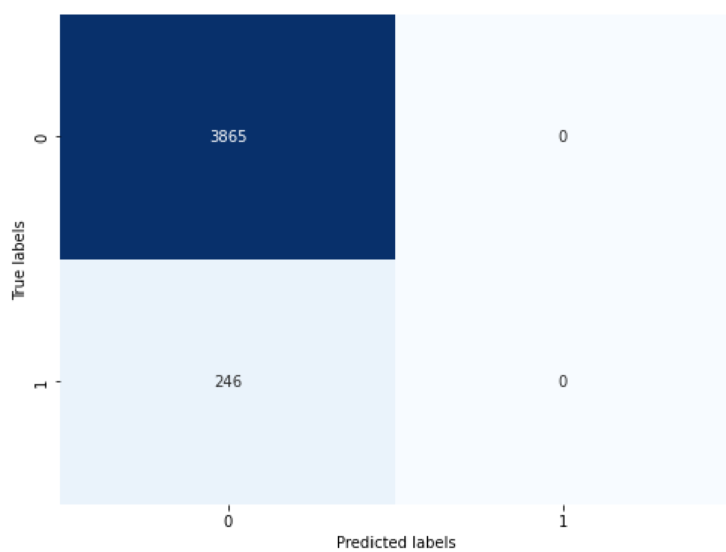
In [ ]:
```
# get rid of rows with NA values
mookie_data_relavent = mookie_data_relavent.dropna()

# logistic regression
X = mookie_data_relavent.drop('is_hit', axis=1)
y = mookie_data_relavent['is_hit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
2)

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9401605448795913
```

In [ ]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      3865
           1       0.00      0.00      0.00       246

    accuracy                           0.94      4111
   macro avg       0.47      0.50      0.48      4111
weighted avg       0.88      0.94      0.91      4111


/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.

/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.

/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
```

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

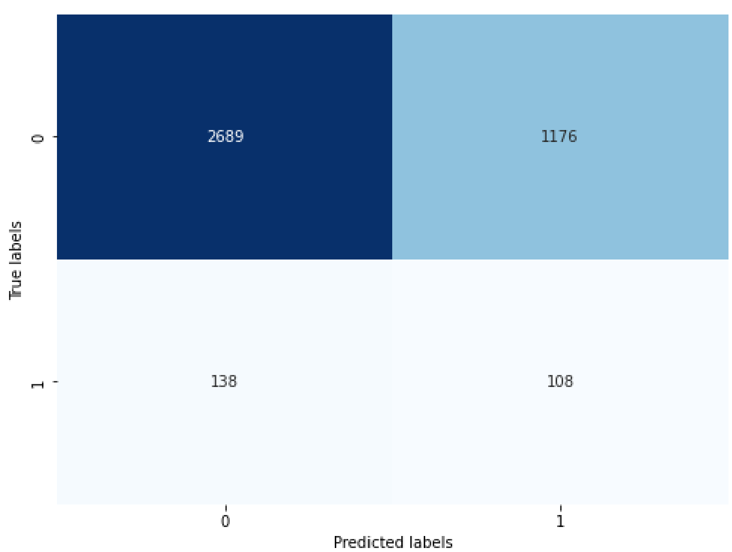


The first logistic regression model is very poor: it predicted that every pitch in the test data would not be a hit. This is most likely due to the fact that the data has many more instances of a pitch resulting in not a hit than a pitch resulting in a hit. I will attempt to use techniques such as weighting and undersampling to see if the model improves.

In [ ]:
```
model = LogisticRegression(class_weight={0: 1, 1: 12})

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.95      0.70      0.80      3865
           1       0.08      0.44      0.14       246

    accuracy                           0.68      4111
   macro avg       0.52      0.57      0.47      4111
weighted avg       0.90      0.68      0.76      4111
```

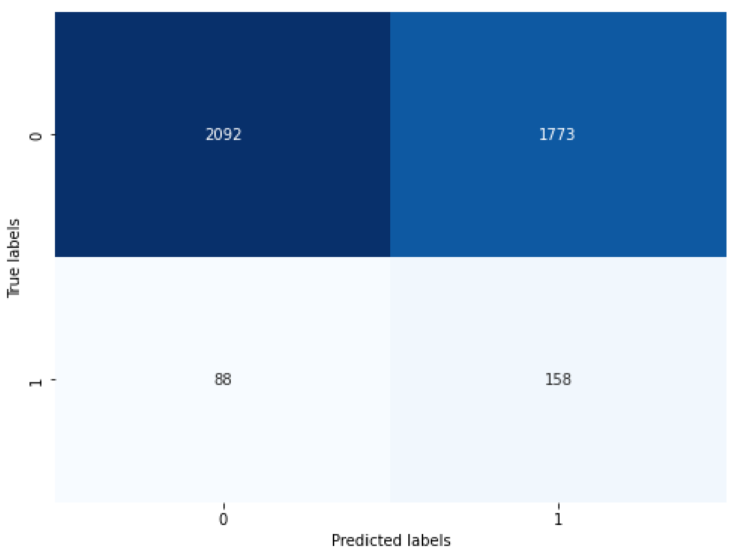
```
/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```



In [ ]:

```
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)

logreg = LogisticRegression()

logreg.fit(X_train_resampled, y_train_resampled)

y_pred = logreg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

report = classification_report(y_test, y_pred)
print(report)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
Accuracy: 0.5473120895159329
              precision    recall  f1-score   support

           0       0.96      0.54      0.69      3865
           1       0.08      0.64      0.15       246

    accuracy                           0.55      4111
   macro avg       0.52      0.59      0.42      4111
weighted avg       0.91      0.55      0.66      4111
```



Random Forest Classification

In [ ]:
```
rf_classifier = RandomForestClassifier(n_estimators=10, random_state=42,
                                       class_weight={0: 1, 1: 12})

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96      3865
           1       0.12      0.02      0.04       246

    accuracy                           0.93      4111
   macro avg       0.53      0.51      0.50      4111
weighted avg       0.89      0.93      0.91      4111
```

In [ ]:
```
rf_classifier = RandomForestClassifier(n_estimators=10, random_state=42,
                                       class_weight={0: 1, 1: 12})

rf_classifier.fit(X_train_resampled, y_train_resampled)

y_pred = rf_classifier.predict(X_test)

print(classification_report(y_test, y_pred))

importances = rf_classifier.feature_importances_
feature_names = X.columns

feature_importances_df = pd.DataFrame({'Feature': feature_names, 'Importance': importanc
es})

feature_importances_df = feature_importances_df.sort_values('Importance', ascending=Fals
e)

n = 10
print(feature_importances_df.head(n))
```

```
              precision    recall  f1-score   support

           0       0.97      0.69      0.81      3865
           1       0.12      0.63      0.20       246

    accuracy                           0.69      4111
   macro avg       0.54      0.66      0.50      4111
weighted avg       0.92      0.69      0.77      4111

          Feature  Importance
1         plate_z    0.467496
0         plate_x    0.389557
17      p_throws_R    0.020176
27       count_3-0    0.016069
6    pitch_type_FF    0.011690
18       count_0-0    0.011518
23       count_1-2    0.008415
14   pitch_type_SL    0.007624
21       count_1-0    0.007421
13   pitch_type_SI    0.007068
```

Boosting

In [ ]:
```
xgb_classifier = XGBClassifier(random_state=42, max_depth = 150)
xgb_classifier.fit(X_train, y_train)

y_pred = xgb_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      3865
           1       0.10      0.04      0.06       246

    accuracy                           0.92      4111
   macro avg       0.52      0.51      0.51      4111
weighted avg       0.89      0.92      0.90      4111
```

In [ ]:
```
xgb_classifier = XGBClassifier(random_state=42, max_depth = 150)
xgb_classifier.fit(X_train_resampled, y_train_resampled)

y_pred = xgb_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.65      0.78      3865
           1       0.12      0.75      0.20       246

    accuracy                           0.65      4111
   macro avg       0.55      0.70      0.49      4111
weighted avg       0.92      0.65      0.74      4111
```

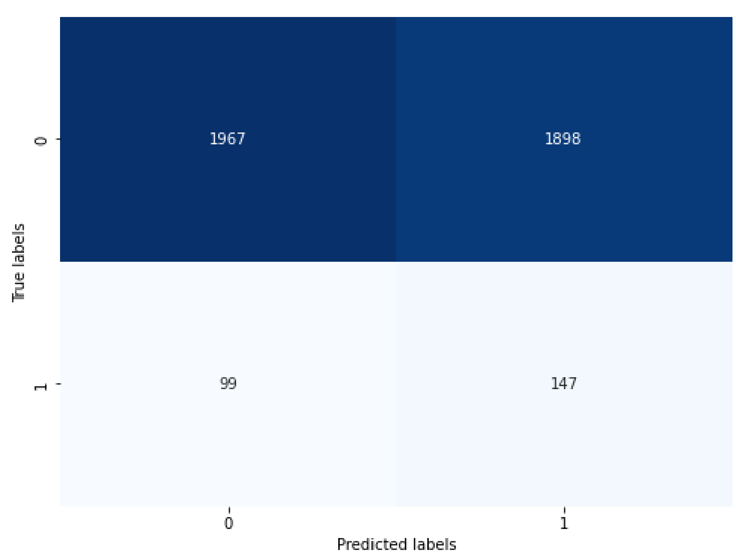
All of the models performed poorly even if techniques such as weighting the classes and undersampling were used. I will now attempt to get rid of some of the features that aren't as important to the models according to the random forest classifier to see if this has any impact on the results.

Pitch Zone Features Only

In [ ]:

```
mookie_data_relavent_filtered = mookie_data[['is_hit','plate_x','plate_z']]

mookie_data_relavent_filtered = mookie_data_relavent_filtered.dropna()

X = mookie_data_relavent_filtered.drop('is_hit', axis=1)
y = mookie_data_relavent_filtered['is_hit']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
2)

rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)

logreg = LogisticRegression()

logreg.fit(X_train_resampled, y_train_resampled)

y_pred = logreg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
Accuracy: 0.5142301143274143
```



In [ ]:
```
coefficients = logreg.coef_[0]
intercept = logreg.intercept_[0]

coefficients_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': coefficients})
intercept_df = pd.DataFrame({'Feature': 'Intercept', 'Coefficient': intercept}, index=
[0])

coefficients_df = pd.concat([intercept_df, coefficients_df])

print(coefficients_df)
```

```
     Feature  Coefficient
0  Intercept    -0.128049
0    plate_x    -0.346249
1    plate_z     0.089453
```

In [ ]:
```
rf_classifier = RandomForestClassifier(n_estimators=10, random_state=42)

rf_classifier.fit(X_train_resampled, y_train_resampled)

y_pred = rf_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

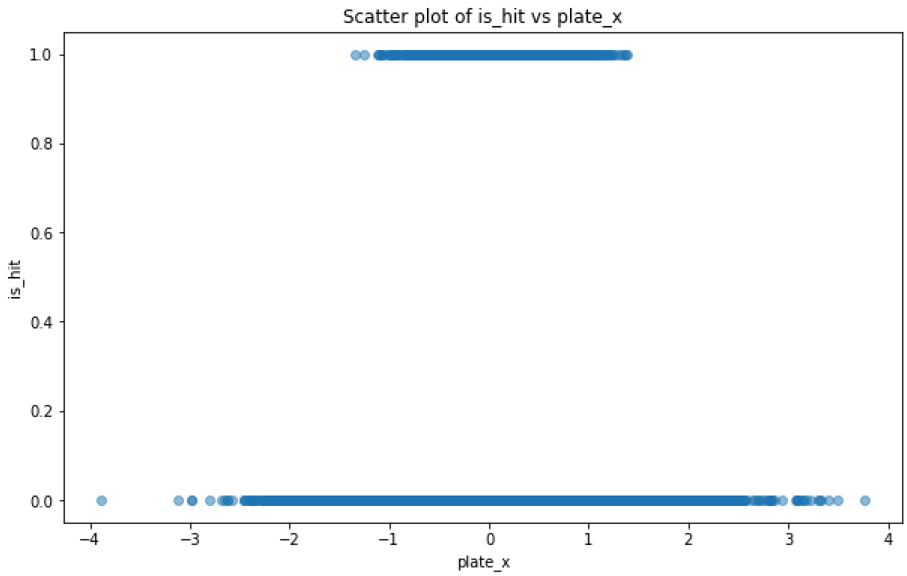
```
              precision    recall  f1-score   support

           0       0.96      0.66      0.78      3865
           1       0.10      0.62      0.18       246

    accuracy                           0.65      4111
   macro avg       0.53      0.64      0.48      4111
weighted avg       0.91      0.65      0.74      4111
```

In [ ]:
```
xgb_classifier = XGBClassifier(random_state=42, max_depth = 150)
xgb_classifier.fit(X_train_resampled, y_train_resampled)

y_pred = xgb_classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.61      0.75      3865
           1       0.10      0.68      0.18       246

    accuracy                           0.62      4111
   macro avg       0.53      0.65      0.46      4111
weighted avg       0.92      0.62      0.72      4111
```

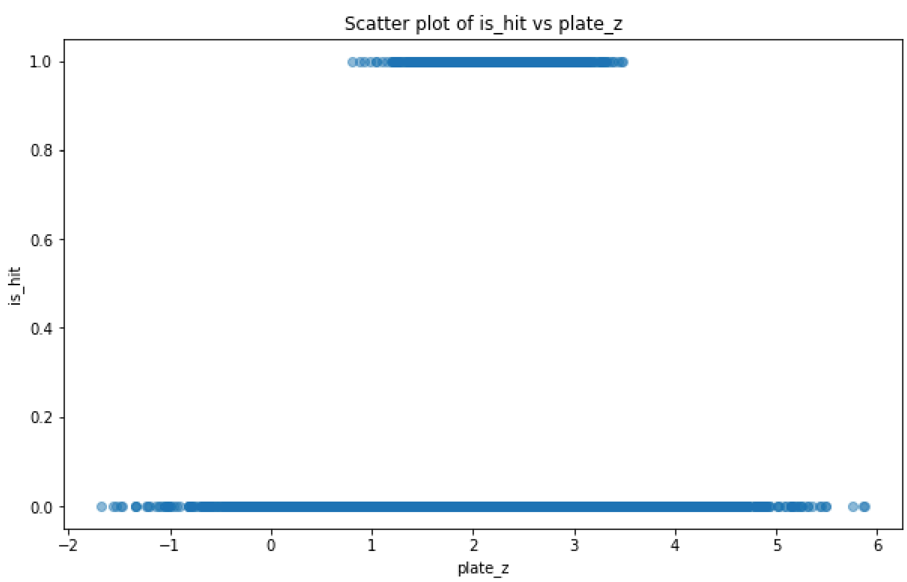
In [ ]:

```
plt.figure(figsize=(10, 6))
plt.scatter(mookie_data_relavent_filtered['plate_x'], mookie_data_relavent_filtered['is_hit'], alpha=0.5)
plt.xlabel('plate_x')
plt.ylabel('is_hit')
plt.title('Scatter plot of is_hit vs plate_x')
plt.show()
```

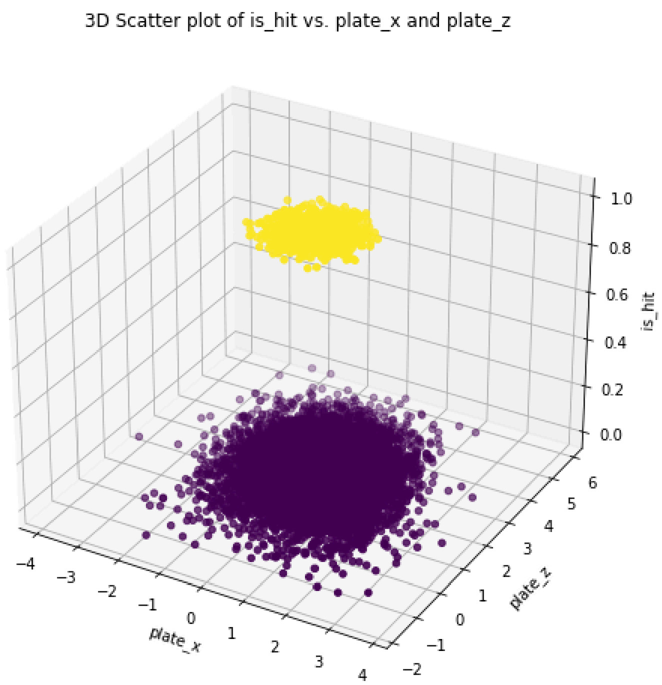


In [ ]:

```
plt.figure(figsize=(10, 6))
plt.scatter(mookie_data_relavent_filtered['plate_z'], mookie_data_relavent_filtered['is_hit'], alpha=0.5)
plt.xlabel('plate_z')
plt.ylabel('is_hit')
plt.title('Scatter plot of is_hit vs plate_z')
plt.show()
```



In [ ]:
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(mookie_data_relavent_filtered['plate_x'], mookie_data_relavent_filtered['plat
e_z'],
mookie_data_relavent_filtered['is_hit'], c=mookie_data_relavent_filtered['is_hit'], cmap
='viridis')

ax.set_xlabel('plate_x')
ax.set_ylabel('plate_z')
ax.set_zlabel('is_hit')
ax.set_title('3D Scatter plot of is_hit vs. plate_x and plate_z')

plt.show()
```



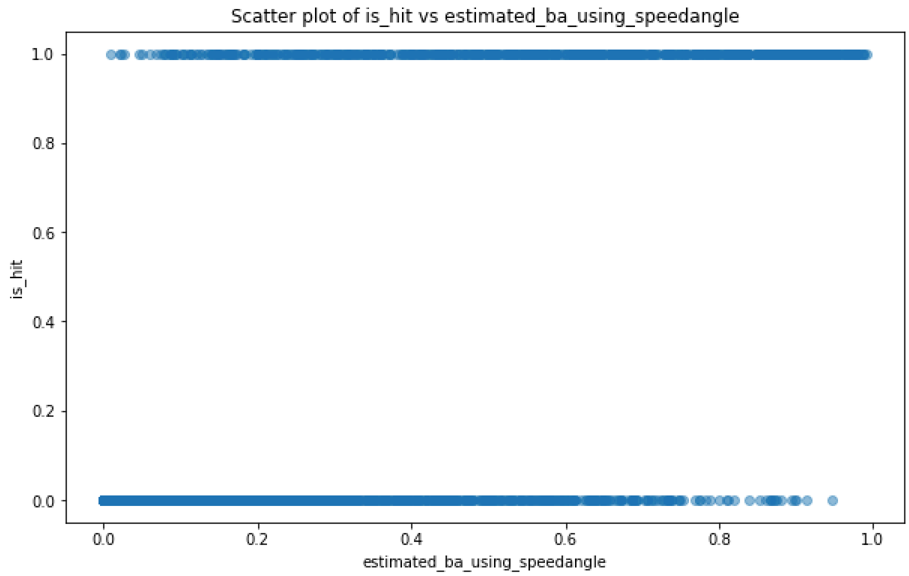
Conclusions

None of the models I created performed as well I wanted them to. The error terms of the models are too high since there is most likely a lot of variance in whether or not a player hits a pitch that I am not able to capture with the data set I'm using. I will now try a different approach; instead of trying to classify each pitch as a hit or not a hit I will run a regression on estimated batting average based on launch angle and exit velocity.

Predicting xBA

In [ ]:
```
# Adjust data sets for algorithms
mookie_exp_ba = mookie_data.dropna(subset=['estimated_ba_using_speedangle', 'plate_x',
'plate_z',
                                            'release_spin_rate'])

shohei_exp_ba = shohei_data.dropna(subset=['estimated_ba_using_speedangle'])

freddie_exp_ba = freddie_data.dropna(subset=['estimated_ba_using_speedangle'])
```

In [ ]:
```
# Estimated BA vs. Actual Hit
plt.figure(figsize=(10, 6))
plt.scatter(mookie_exp_ba['estimated_ba_using_speedangle'], mookie_exp_ba['is_hit'], alp
ha=0.5)
plt.xlabel('estimated_ba_using_speedangle')
plt.ylabel('is_hit')
plt.title('Scatter plot of is_hit vs estimated_ba_using_speedangle')
plt.show()
```

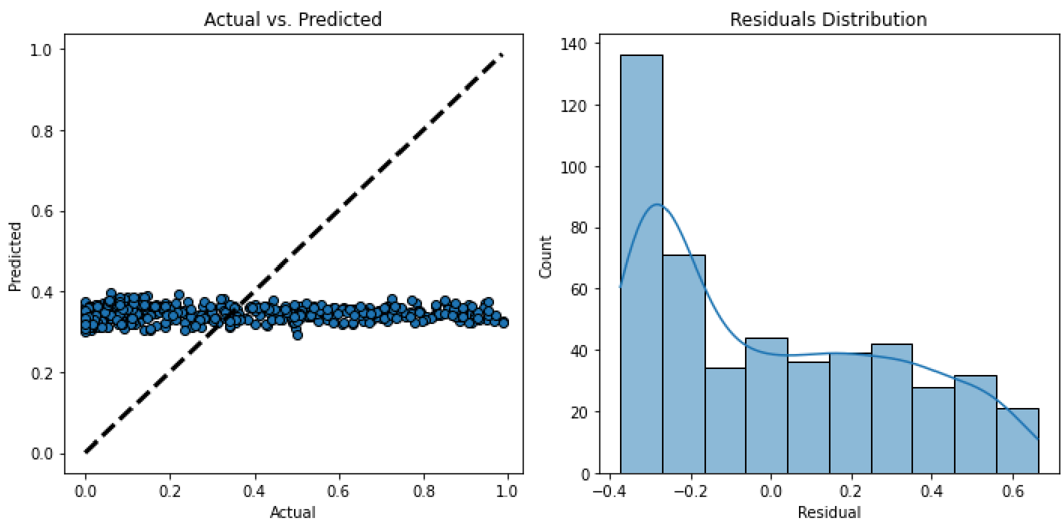


xBA Models

In [ ]:
```
# Linear Regression Model
X = mookie_exp_ba[['plate_x', 'plate_z']]
Y = mookie_exp_ba['estimated_ba_using_speedangle']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

model = LinearRegression()

model.fit(X_train, Y_train)

Y_pred = model.predict(X_test)

print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)

print('Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred))
print('Coefficient of determination (R^2):', r2_score(Y_test, Y_pred))
```

```
Coefficients: [ 0.02582453 -0.022569  ]
Intercept: 0.3928991544978425
Mean squared error (MSE): 0.09097230624436538
Coefficient of determination (R^2): 0.004370302465309894
```

In [ ]:
```
# Plot Actual vs. Predicted values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(Y_test, Y_pred, edgecolors=(0, 0, 0))
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], 'k--', lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted')

# Plot Residuals
residuals = Y_test - Y_pred
plt.subplot(1, 2, 2)
sns.histplot(residuals, kde=True)
plt.xlabel('Residual')
plt.title('Residuals Distribution')

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



The models perform poorly. I will try to get them to perform better by not using the x and z coordinates of the pitch but use those coordinates to create a new variable that represents the pitch's distance from the center of the strike zone.

Models with Distance Variable

In [ ]:
```
# Transform x and z positions into a more relavent variable
def add_distance_from_center(df, plate_x_col='plate_x', plate_z_col='plate_z'):
    # Center of the strike zone
    center_plate_x = 0
    center_plate_z = 2.5

    # Calculate the distance from the center of the strike zone
    df['distance_from_center'] = np.sqrt((df[plate_x_col] - center_plate_x)**2 + (df[pla
te_z_col] - center_plate_z)**2)

    return df
```

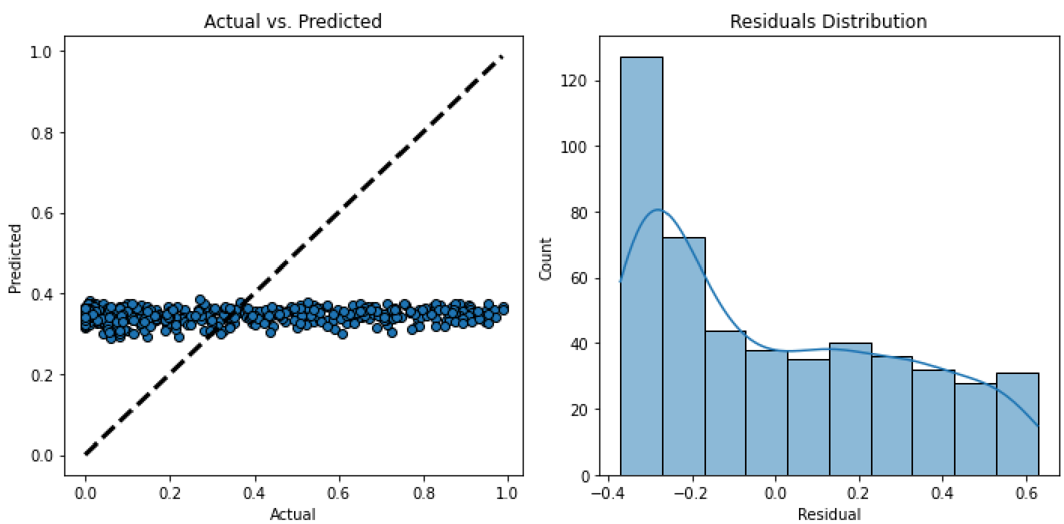
In [ ]:
```
mookie_exp_ba = add_distance_from_center(mookie_exp_ba)
```

```
/var/folders/tk/z92pmxqn68575jr94x116gs00000gn/T/ipykernel_94053/2851089240.py:8: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

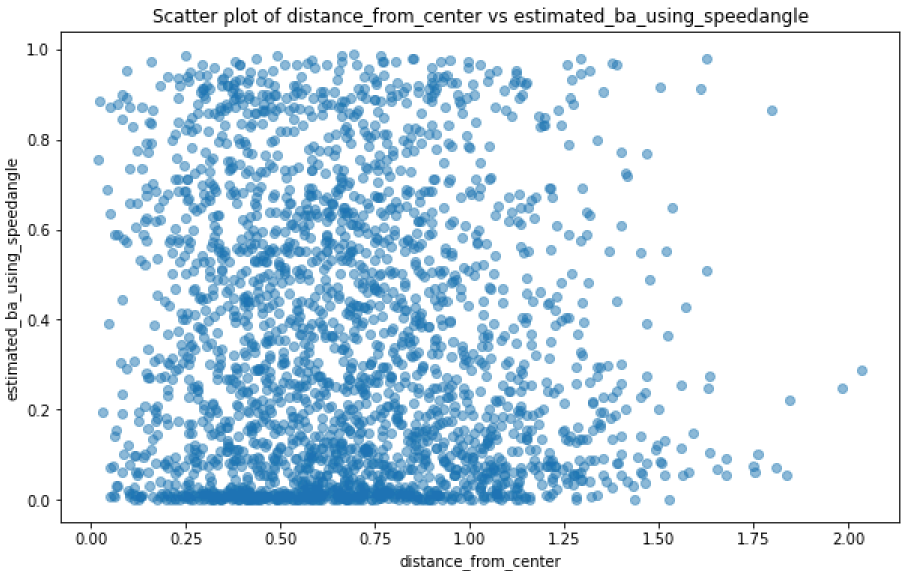
In [ ]:
```
# Linear Regression Model with new variable
X = mookie_exp_ba[['distance_from_center','release_spin_rate']]
Y = mookie_exp_ba['estimated_ba_using_speedangle']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

model = LinearRegression()

model.fit(X_train, Y_train)

Y_pred = model.predict(X_test)

print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)

print('Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred))
print('Coefficient of determination (R^2):', r2_score(Y_test, Y_pred))
```

```
Coefficients: [-4.85892865e-02 -2.10935295e-05]
Intercept: 0.4250833806579222
Mean squared error (MSE): 0.09037313562590998
Coefficient of determination (R^2): 0.010927815254114015
```

In [ ]:
```python
# Plot Actual vs. Predicted values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(Y_test, Y_pred, edgecolors=(0, 0, 0))
plt.plot([min(Y_test), max(Y_test)], [min(Y_test), max(Y_test)], 'k--', lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted')

# Plot Residuals
residuals = Y_test - Y_pred
plt.subplot(1, 2, 2)
sns.histplot(residuals, kde=True)
plt.xlabel('Residual')
plt.title('Residuals Distribution')

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



In [ ]:
```
# Visualize pitch's distance from center and expected BA
plt.figure(figsize=(10, 6))
plt.scatter(mookie_exp_ba['distance_from_center'], mookie_exp_ba['estimated_ba_using_spe
edangle'], alpha=0.5)
plt.xlabel('distance_from_center')
plt.ylabel('estimated_ba_using_speedangle')
plt.title('Scatter plot of distance_from_center vs estimated_ba_using_speedangle')
plt.show()
```



In [ ]:
```
# Random forest model
X = mookie_exp_ba[['plate_x', 'plate_z']]
Y = mookie_exp_ba['estimated_ba_using_speedangle']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

rf_model = RandomForestRegressor(random_state=0)

rf_model.fit(X_train, Y_train)

Y_pred_rf = rf_model.predict(X_test)

print('Random Forest Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred_rf))
print('Random Forest Coefficient of determination (R^2):', r2_score(Y_test, Y_pred_rf))
```

```
Random Forest Mean squared error (MSE): 0.10943123030502605
Random Forest Coefficient of determination (R^2): -0.19765000171346547
```

In [ ]:
```
# Boosting Model
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', random_state=0)

xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)

print('XGBoost Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred_xgb))
print('XGBoost Coefficient of determination (R^2):', r2_score(Y_test, Y_pred_xgb))
```

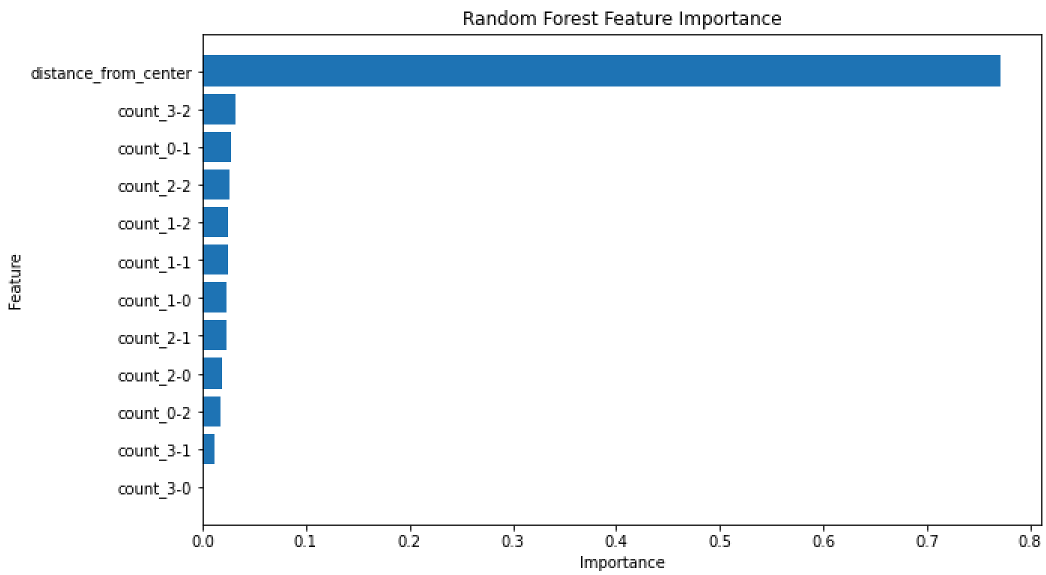
```
XGBoost Mean squared error (MSE): 0.11381750235633732
XGBoost Coefficient of determination (R^2): -0.2456547505874931
```

Even with the newly defined distance variable the models are still unable to perform well. I will now add another predictor that is relavent to predicting the result of a pitch and at-bat: the count before the pitch is thrown.

In [ ]:
```
# Include count before pitch in models
relevant_columns = ['distance_from_center', 'count', 'estimated_ba_using_speedangle']
mookie_relevant = mookie_exp_ba[relevant_columns]

mookie_exp_ba_encoded = pd.get_dummies(mookie_relevant, columns=['count'], drop_first=True)

X = mookie_exp_ba_encoded.drop('estimated_ba_using_speedangle', axis=1)
Y = mookie_exp_ba_encoded['estimated_ba_using_speedangle']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

rf_model = RandomForestRegressor(random_state=0)

rf_model.fit(X_train, Y_train)

Y_pred_rf = rf_model.predict(X_test)

print('Random Forest Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred_rf))
print('Random Forest Coefficient of determination (R^2):', r2_score(Y_test, Y_pred_rf))
```

```
Random Forest Mean squared error (MSE): 0.12763405420076013
Random Forest Coefficient of determination (R^2): -0.396867464673073
```

In [ ]:
```
importances = rf_model.feature_importances_
feature_names = X.columns

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance
s})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=Fal
se)

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Random Forest Feature Importance')
plt.gca().invert_yaxis()
plt.show()
```



In [ ]:
```
# Boosting model with count as factor variable
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', random_state=0)

xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)

print('XGBoost Mean squared error (MSE):', mean_squared_error(Y_test, Y_pred_xgb))
print('XGBoost Coefficient of determination (R^2):', r2_score(Y_test, Y_pred_xgb))
```

```
XGBoost Mean squared error (MSE): 0.12304487171266282
XGBoost Coefficient of determination (R^2): -0.3466419997905794
```

Conclusions

Including the count before the pitch did not help improve the performance of the models. There is most likely too much variability on a pitch-by-pitch and even at-bat-by-at-bat basis to create a model that is able to predict results at an acceptable confidence level. I could have tried to add more predictors but if even distance from the center of the plate and the count don't have any predictive power I highly doubt any other predictors will since it is known in the baseball community that those two variables are important in how well a batter performs. I will now look at hitting streak to see if that has any predictive power (slumping vs. on fire idea in sports).

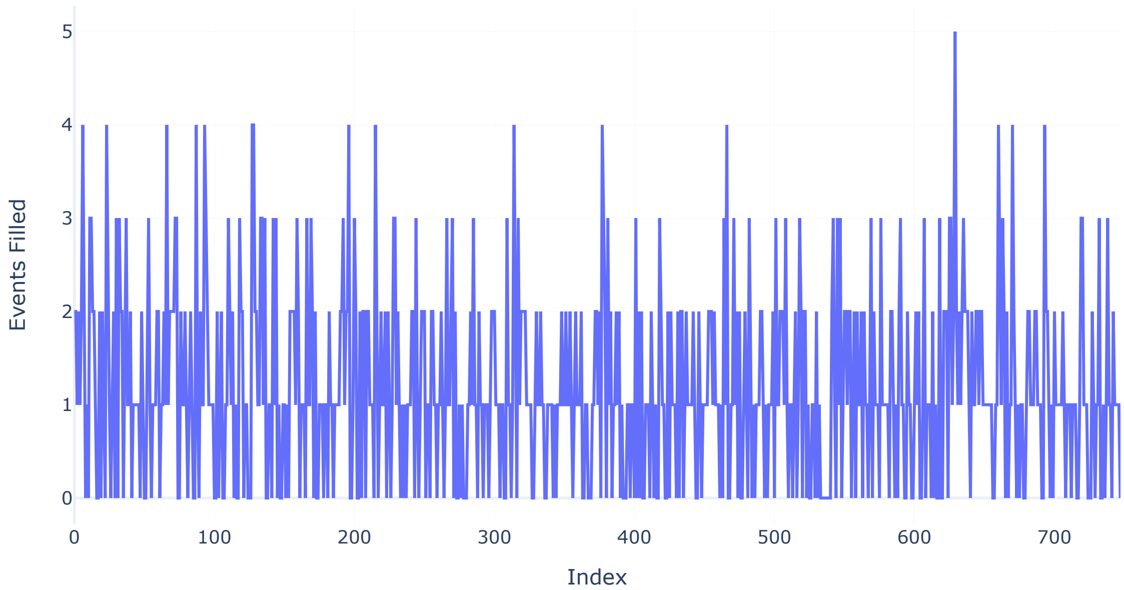
Hitting Streak and Hit or No Hit

In [ ]:
```
def calculate_streak(column):
    streak = [0] * len(column)
    for i in range(1, len(column)):
        if column[i] == column[i - 1]:
            streak[i] = streak[i - 1] + 1
        else:
            streak[i] = 1
    return streak
```

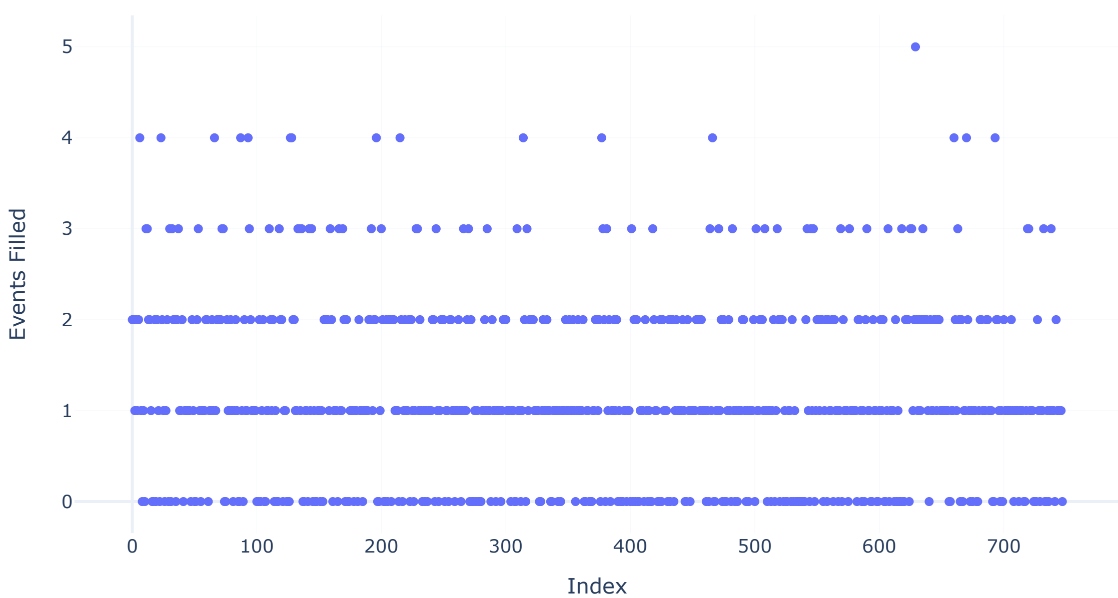
In [ ]:
```
mookie_data = mookie_data.reset_index(drop=True)

mookie_data['streak'] = calculate_streak(mookie_data['is_hit'])

# Function to calculate hitting or no-hitting streaks per game
def calculate_streak_per_game(column, game_ids):
    streak = [0] * len(column)
    count = {game_id: 0 for game_id in game_ids}
    for i in range(1, len(column)):
        if column[i] == column[i - 1]:
            count[game_ids[i]] += 1
        else:
            count[game_ids[i]] = 0
        streak[i] = count[game_ids[i]] + 1 if column[i] == column[i - 1] else 1
    return streak

game_ids = mookie_data['game_pk']

mookie_data['streak'] = calculate_streak_per_game(mookie_data['is_hit'], game_ids)
```

In [ ]:
```
# Visualize hits in a game as time series
mookie_games['index'] = mookie_games.index

# Line plot
fig = px.line(mookie_games,
              x='index',
              y='events_filled',
              title='Events Filled vs Index: Mookie Betts',
              labels={'events_filled': 'Events Filled', 'index': 'Index'},
              template='plotly_white')

fig.update_layout(
    xaxis_title="Index",
    yaxis_title="Events Filled",
)

fig.write_html('plot.html')

fig.show()

# Scatter plot
fig = px.scatter(mookie_games,
                 x='index',
                 y='events_filled',
                 title='Events Filled vs Index: Mookie Betts',
                 labels={'events_filled': 'Events Filled', 'index': 'Index'},
                 template='plotly_white')

fig.update_layout(
    xaxis_title="Index",
    yaxis_title="Events Filled",
)

fig.write_html('plot.html')

fig.show()
```

Events Filled vs Index: Mookie Betts



Events Filled vs Index: Mookie Betts



In [ ]:
```
# Function to calculate hitting streaks across games
def calculate_streak(df):
    streak = []
    count = 0
    for i in range(len(df)):
        if df['events_filled'].iloc[i] > 0:
            if count >= 0:
                count += 1
            else:
                count = 1
        else:
            if count <= 0:
                count -= 1
            else:
                count = -1
        streak.append(count)
    return streak

# Apply the function to the DataFrame
mookie_games['streak'] = calculate_streak(mookie_games)
```

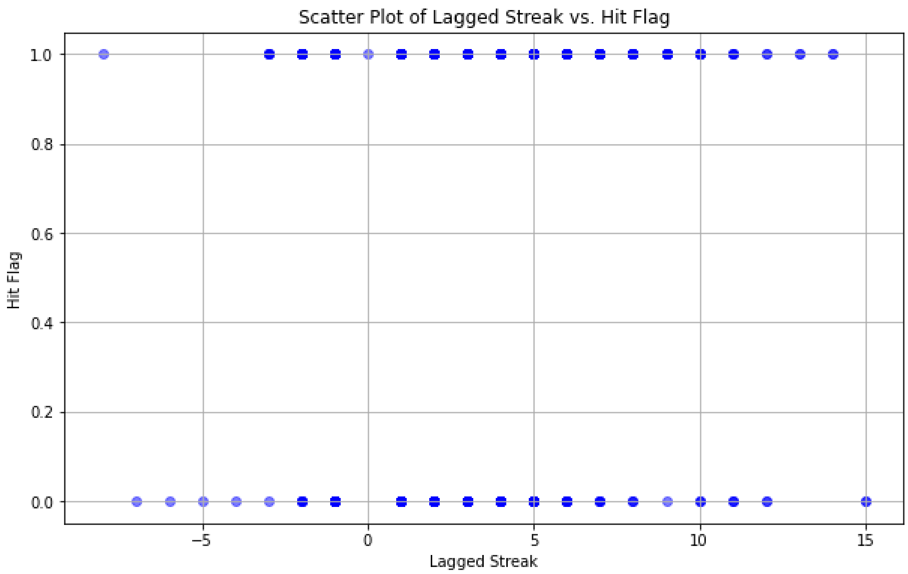
In [ ]:
```
mookie_games['hit_flag'] = mookie_games['events_filled'].apply(lambda x: 1 if x > 0 else
0)
```

In [ ]:
```
# Logistic regression model
mookie_games['lagged_streak'] = mookie_games['streak'].shift(1)
mookie_games['lagged_streak'].fillna(0, inplace=True)

X = mookie_games[['lagged_streak']]
y = mookie_games['hit_flag']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
2)

logit_model = LogisticRegression()
logit_model.fit(X_train, y_train)

y_pred = logit_model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

mookie_games['predicted_hit_flag'] = logit_model.predict(X)
```

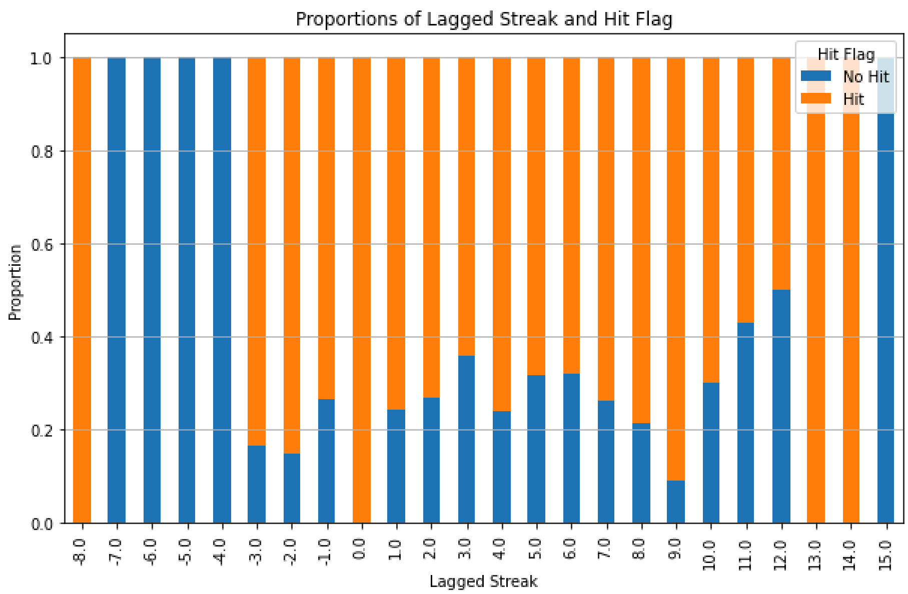
```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        58
           1       0.74      1.00      0.85       167

    accuracy                           0.74       225
   macro avg       0.37      0.50      0.43       225
weighted avg       0.55      0.74      0.63       225

[[  0  58]
 [  0 167]]
```

```
/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.

/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.

/Users/jedihernandez/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarnin
g:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
```

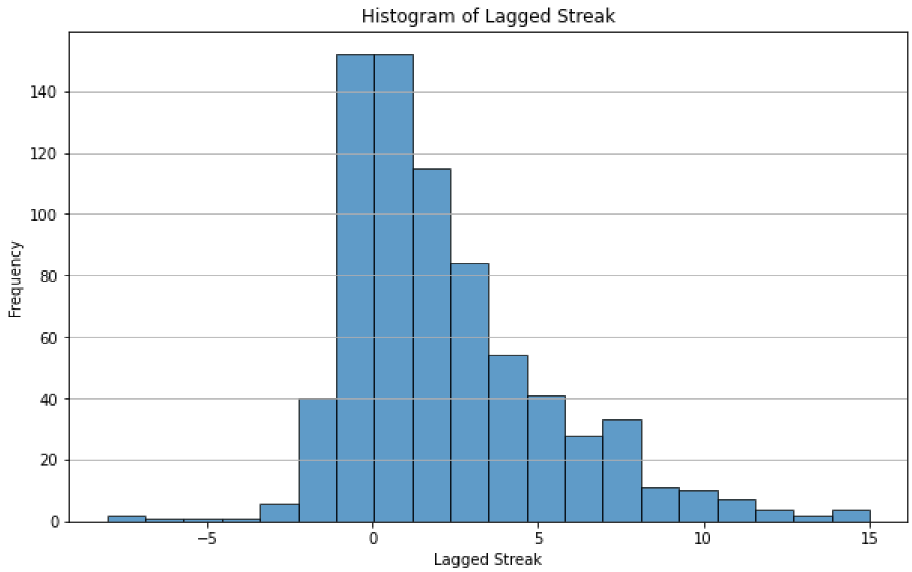
In [ ]:
```
# Visualize hitting streak and hit or no hit next game
plt.figure(figsize=(10, 6))
plt.scatter(mookie_games['lagged_streak'], mookie_games['hit_flag'], color='b', alpha=0.
5)
plt.title('Scatter Plot of Lagged Streak vs. Hit Flag')
plt.xlabel('Lagged Streak')
plt.ylabel('Hit Flag')
plt.grid(True)
plt.show()
```



In [ ]:
```
# Look at proportions of lagged streak and hit or no hit
proportions = mookie_games.groupby(['lagged_streak', 'hit_flag']).size().unstack().apply
(lambda x: x / x.sum(), axis=1)

proportions.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Proportions of Lagged Streak and Hit Flag')
plt.xlabel('Lagged Streak')
plt.ylabel('Proportion')
plt.legend(title='Hit Flag', labels=['No Hit', 'Hit'], loc='upper right')
plt.grid(axis='y')
plt.show()
```



In [ ]:
```
# Create a histogram to see total counts
plt.figure(figsize=(10, 6))
plt.hist(mookie_games['lagged_streak'], bins=20, edgecolor='black', alpha=0.7)
plt.title('Histogram of Lagged Streak')
plt.xlabel('Lagged Streak')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```



Conclusions

Using the hitting streak to attempt to predict hit or no hit the next game was not sucessful. The models were poor and inspecting the data shows that there isn't any discernible pattern but the variables look uncorrelated. Perhaps there is a way to develop a model that accurately predicts whether or not a player gets a hit in a future game but based on the work I have done here I conclude that it is much more probable that there is too much variability that is not contained in the data and/or random chance that makes it impossible to develop a model that will make predictions with enough accuracy to develop a betting strategy from.