index

May 1, 2022

Data Analytics on Player Performance in Major League Baseball

Chris Emm

[1]:
```
import sqlite3
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.ticker as mticker
import requests
from bs4 import BeautifulSoup
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from scipy.stats import pearsonr
import statsmodels.formula.api as smf
import seaborn
from sklearn.preprocessing import OneHotEncoder
import warnings

pd.set_option('display.max_columns', None)
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/compat/pandas.py:65:
FutureWarning: pandas.Int64Index is deprecated and will be removed from pandas
in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import Int64Index as NumericIndex
```

0.0.1 Introduction

Baseball is a game of scoring runs. There's a reason that the team with the most runs at the end of a game wins. Major Leage Baseball (MLB), especially in the past 20 years has seen an uptick of scoring, as the game has become more and more about offensive firepower rather than pitchers completely dominating the hitters. A team's front office and everyone that is included into the decision making behind roster formations need to be able to analyze player performance and determine which players will score them the most runs, and in effect, help them win the most games. In this project, we will analyze which offensive metrics are most closely related to scoring runs, using team data between 2011-2021. Then, based on our findings, we will then create a predictive model that will extrapolate which players are most likely to perform well with regards to the metrics we deem to important in driving in runs.

0.0.2 Part I: Scraping Team Data for 2000-2021 Seasons

The first thing we are going to do is analyze a variety of offensive metrics and their relation to producing runs on offense. In order to do this, we will need to scrape team data from FanGraphs (https://www.fangraphs.com/). We will gather basic, advanced, and batted ball data that each team accumulated over each season for the last decade. Below are two functions that scrape the data from the website.

The following function scrapes the table that is located at the specified url, and creates a dataframe using pandas from the table that is scraped. The additional year and team arguments allow us to add respective columns based on which team each row is for.

[2]:
```
def scraping_FanGraphs(url, year, team):
    # Extracting text from webpage
    html = requests.get(url).text

    # Parsing the text into html code
    soup = BeautifulSoup(html,"html.parser")

    # Finding the table in the html code - we are searching by the id of the␣
↪table
    table = soup.find("table", attrs={"class": "rgMasterTable"})


    table_data = table.tbody.find_all("tr")

    dataset = []
    for tr in table_data:
        temp = ()
        for td in tr.find_all("td"):
            if '\xa0' in td.text:
                temp += ('0.0',)
            else:
                temp += (td.text,)
        dataset.append(temp)

    stats = pd.DataFrame(data = dataset)
    stats = stats.replace(to_replace=" NULL",value=0)

    table_header = table.thead.find_all("tr")
    columns = []
    count = 0
    for tr in table_header:
        if count == 1:
            th = tr.find_all("th")
            for a in th:
                columns.append(a.text)
        count = 1
```

```python
    stats.columns = columns
    stats = stats.assign(Year = year)
    if team != 'None':
        stats = stats.assign(Team = team)

    return stats
```

The function below simply compiles a list of urls based on which FanGraphs page we want to visit. Since the basic, advanced, and batted ball statistics are on separate urls, we have an argument, stat, which determines which url we are looking to scrape from. This function will be used to create urls for all 30 MLB teams for the years that are specified (2011-2021). The page argument is used because some teams have too many players to fit on one page, so the remaining are placed on separate pages. As you can see, we wil use this function for both team and player scraping.

[3]:
```python
def get_urls(team, year, page, stat):

######################################################################
#                       Player Stats Urls                            #
######################################################################

    if stat == 'player_standard':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=0&season=' + str(year[1]) +␣
 ↪'&month=0&season1=' + str(year[0]) + '&ind=1' \
            '&team='+ str(team)␣
 ↪+'&rost=0&age=0&filter=&players=0&startdate=&enddate=&page=' + str(page) +␣
 ↪'_50'

    if stat == 'player_advanced':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=1&season=' + str(year[1]) +␣
 ↪'&month=0&season1=' + str(year[0]) + '&ind=1' \
            '&team='+ str(team)␣
 ↪+'&rost=0&age=0&filter=&players=0&startdate=&enddate=&page=' + str(page) +␣
 ↪'_50'

    if stat == 'player_batted':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=2&season=' + str(year[1]) +␣
 ↪'&month=0&season1=' + str(year[0]) + '&ind=1' \
            '&team='+ str(team)␣
 ↪+'&rost=0&age=0&filter=&players=0&startdate=&enddate=&page=' + str(page) +␣
 ↪'_50'

    if stat == 'player_statcast':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
```

```
            '&lg=all&qual=0&type=24&season=' + str(year[1]) +␣
↪'&month=0&season1=' + str(year[0]) + '&ind=1' \
            '&team='+ str(team)␣
↪+'&rost=0&age=0&filter=&players=0&startdate=&enddate=&page=' + str(page) +␣
↪'_50'

    if stat == 'player_plate_discipline':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=5&season=' + str(year[1]) +␣
↪'&month=0&season1=' + str(year[0]) + '&ind=1' \
            '&team='+ str(team)␣
↪+'&rost=0&age=0&filter=&players=0&startdate=&enddate=&page=' + str(page) +␣
↪'_50'

######################################################################
#                        Team Stats Urls                            #
######################################################################

    if stat == 'team':
        url = 'https://www.fangraphs.com/leaders.aspx?
↪pos=all&stats=bat&lg=all&qual=0&type=0&season=' + str(year) +  \
            '&month=0&season1=' + str(year) +␣
↪'&ind=0&team=0,ts&rost=0&age=0&filter=&players=0&startdate=' + str(year) +␣
↪'-01-01&enddate=' + str(year) + '-12-31'

    if stat == 'team_advanced':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=1&season=' + str(year) + '&month=0&season1=' +␣
↪str(year) + '&ind=0&team=0,'\
            'ts&rost=0&age=0&filter=&players=0&startdate=' + str(year) +␣
↪'-01-01&enddate=' + str(year) + '-12-31'

    if stat == 'team_batted':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=2&season=' + str(year) + '&month=0&season1=' +␣
↪str(year) + '&ind=0&'\
            'team=0,ts&rost=0&age=0&filter=&players=0&startdate=' + str(year) +␣
↪'-01-01&enddate=' + str(year) + '-12-31'

    if stat == 'team_statcast':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=24&season=' + str(year) + '&month=0&season1='␣
↪+ str(year) + '&ind=0' \
            '&team=0,ts&rost=0&age=0&filter=&players=0&startdate=' + str(year)␣
↪+ '-01-01&enddate=' + str(year) + '-12-31'
```

```
    if stat == 'team_plate_discipline':
        url = 'https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat' \
            '&lg=all&qual=0&type=5&season=' + str(year) + '&month=0&season1=' +␣
 ↪str(year) + '&ind=0' \
            '&team=0,ts&rost=0&age=0&filter=&players=0&startdate=' + str(year)␣
 ↪+ '-01-01&enddate=' + str(year) + '-12-31'
    return url
```

Scraping Team Data From Fangraphs Here we are actually compiling the web scrape results and merging all resulting dataframes into one overall dataframe called team_batting.

[4]:
```
years = [i for i in range(2000,2022)]

######################################################################
#               Creating a Dataframe for Team Stats                  #
######################################################################

count = 0
for year in years:
    # Since we are scraping for team, we don't need to specify a team or page␣
 ↪(those are arguments for player scraping)
    url = get_urls('None', year, 'None', 'team')
    if count == 0:
        team_batting = scraping_FanGraphs(url, year, 'None')
        # In 2011, Miami Marlins were the Florida Marlins (they changed to␣
 ↪Miami in 2012)
        team_batting['Team'] = team_batting['Team'].replace({'FLA':'MIA'},␣
 ↪regex = True)
        count = 1
    else:
        team_batting = pd.concat([team_batting, scraping_FanGraphs(url, year,␣
 ↪'None')])


team_batting = team_batting.drop_duplicates()
team_batting = team_batting.reset_index(drop=True)
team_batting = team_batting[['Year', 'Team', 'AB', 'PA', 'AVG', 'H', '1B',␣
 ↪'2B', \
                             '3B', 'HR', 'R', 'RBI', 'BB', 'IBB', 'SO', 'HBP', \
                             'SF', 'SH', 'GDP', 'SB', 'CS']]

######################################################################
#            Adding Advanced Batting Stats to Dataframe              #
######################################################################

count = 0
```

```
for year in years:
    # Since we are scraping for team, we don't need to specify a team or page␣
 ↪(those are arguments for player scraping)
    url = get_urls('None', year, 'None', 'team_advanced')
    if count == 0:
        team_advanced_batting = scraping_FanGraphs(url, year, 'None')
        # In 2011, Miami Marlins were the Florida Marlins (they changed to␣
 ↪Miami in 2012)
        team_batting['Team'] = team_batting['Team'].replace({'FLA':'MIA'},␣
 ↪regex = True)
        count = 1
    else:
        team_advanced_batting = pd.concat([team_advanced_batting,␣
 ↪scraping_FanGraphs(url, year, 'None')])


team_advanced_batting = team_advanced_batting.drop_duplicates()
team_advanced_batting = team_advanced_batting.reset_index(drop=True)
team_advanced_batting = team_advanced_batting[['Year', 'Team', 'PA', 'BB%',␣
 ↪'K%', \
                                               'BB/K', 'AVG', 'OBP', 'SLG',␣
 ↪'OPS', 'ISO', \
                                               'Spd', 'BABIP', 'UBR', 'wGDP',␣
 ↪'wSB', 'wRC', \
                                               'wRAA', 'wOBA', 'wRC+']]

# Merge data into team batting dataframe
team_batting = pd.merge(team_batting, team_advanced_batting, on = ['Year',␣
 ↪'Team', 'PA', 'AVG'])

######################################################################
#             Adding Batted Ball Stats to Dataframe                  #
######################################################################

count = 0
for year in years:
    # Since we are scraping for team, we don't need to specify a team or page␣
 ↪(those are arguments for player scraping)
    url = get_urls('None', year, 'None', 'team_batted')
    if count == 0:
        team_advanced_batting = scraping_FanGraphs(url, year, 'None')
        # In 2011, Miami Marlins were the Florida Marlins (they changed to␣
 ↪Miami in 2012)
        team_batting['Team'] = team_batting['Team'].replace({'FLA':'MIA'},␣
 ↪regex = True)
        count = 1
```

```
    else:
        team_advanced_batting = pd.concat([team_advanced_batting,␣
 ↪scraping_FanGraphs(url, year, 'None')])


team_advanced_batting = team_advanced_batting.drop_duplicates()
team_advanced_batting = team_advanced_batting.reset_index(drop=True)
team_advanced_batting = team_advanced_batting[['Year', 'Team', 'BABIP', 'GB/
 ↪FB', \
                                               'LD%', 'GB%', 'FB%', 'IFFB%',␣
 ↪'HR/FB', \
                                               'IFH', 'IFH%', 'BUH', 'BUH%',␣
 ↪'Pull%', \
                                               'Cent%', 'Oppo%', 'Soft%',␣
 ↪'Med%', 'Hard%']]

# Merge data into team batting dataframe
team_batting = pd.merge(team_batting, team_advanced_batting, on = ['Year',␣
 ↪'Team', 'BABIP'])

######################################################################
#                Adding Statcast Data to Dataframe                   #
######################################################################

count = 0
for year in years:
    # Since we are scraping for team, we don't need to specify a team or page␣
 ↪(those are arguments for player scraping)
    url = get_urls('None', year, 'None', 'team_statcast')
    if count == 0:
        team_advanced_batting = scraping_FanGraphs(url, year, 'None')
        # In 2011, Miami Marlins were the Florida Marlins (they changed to␣
 ↪Miami in 2012)
        team_batting['Team'] = team_batting['Team'].replace({'FLA':'MIA'},␣
 ↪regex = True)
        count = 1
    else:
        team_advanced_batting = pd.concat([team_advanced_batting,␣
 ↪scraping_FanGraphs(url, year, 'None')])


team_advanced_batting = team_advanced_batting.drop_duplicates()
team_advanced_batting = team_advanced_batting.reset_index(drop=True)
team_advanced_batting = team_advanced_batting[['Year', 'Team', 'EV', 'LA',␣
 ↪'Barrel%', 'HardHit%']]
```

```
# Merge data into team batting dataframe
team_batting = pd.merge(team_batting, team_advanced_batting, on = ['Year',␣
↪'Team'])

######################################################################
#           Adding Plate Discipline Data to Dataframe                #
######################################################################

count = 0
for year in years:
    # Since we are scraping for team, we don't need to specify a team or page␣
↪(those are arguments for player scraping)
    url = get_urls('None', year, 'None', 'team_plate_discipline')
    if count == 0:
        team_advanced_batting = scraping_FanGraphs(url, year, 'None')
        # In 2011, Miami Marlins were the Florida Marlins (they changed to␣
↪Miami in 2012)
        team_batting['Team'] = team_batting['Team'].replace({'FLA':'MIA'},␣
↪regex = True)
        count = 1
    else:
        team_advanced_batting = pd.concat([team_advanced_batting,␣
↪scraping_FanGraphs(url, year, 'None')])


team_advanced_batting = team_advanced_batting.drop_duplicates()
team_advanced_batting = team_advanced_batting.reset_index(drop=True)
team_advanced_batting = team_advanced_batting[['Year', 'Team', 'O-Swing%',␣
↪'Z-Swing%', 'Swing%', \
                                               'O-Contact%', 'Z-Contact%',␣
↪'Contact%', 'Zone%', \
                                               'F-Strike%', 'SwStr%', 'CStr%',␣
↪'CSW%']]

# Merge data into team batting dataframe
team_batting = pd.merge(team_batting, team_advanced_batting, on = ['Year',␣
↪'Team'])

######################################################################
#              Adding Wins and Losses to Dataframe                   #
######################################################################

teams_table = pd.read_csv('tables/Teams.csv')
teams_table = teams_table[teams_table.yearID > 1999]

teams_table = teams_table.rename(columns = {'yearID':'Year','franchID':'Team'})
```

```
# Taking only the necessary columns
teams_table = teams_table[['Year', 'Team', 'W', 'L']]

data = []
for team_index, team_row in teams_table.iterrows():
    for my_team_index, my_team_row in team_batting.iterrows():
        if my_team_row['Team'] == team_row['Team'] and my_team_row['Year'] ==␣
 ↪team_row['Year']:
            team = list(my_team_row)
            team.append(team_row['W'])
            team.append(team_row['L'])
            team = tuple(team)
            data.append(team)

# Creating a dataframe from the list of tuples above
team_batting = pd.DataFrame(data, columns=['Year', 'Team', 'AB', 'PA', 'AVG', \
                                           'H', '1B', '2B', '3B', 'HR', 'R',␣
 ↪'RBI',\
                                           'BB', 'IBB', 'SO', 'HBP', 'SF',␣
 ↪'SH', 'GDP',\
                                           'SB', 'CS', 'BB%', 'K%', 'BB/K',␣
 ↪'OBP', 'SLG',\
                                           'OPS', 'ISO', 'Spd', 'BABIP', 'UBR',␣
 ↪'wGDP', \
                                           'wSB', 'wRC', 'wRAA', 'wOBA',␣
 ↪'wRC+', 'GB/FB',\
                                           'LD%', 'GB%', 'FB%', 'IFFB%', 'HR/
 ↪FB', 'IFH', \
                                           'IFH%', 'BUH', 'BUH%', 'Pull%',␣
 ↪'Cent%', 'Oppo%',\
                                           'Soft%', 'Med%', 'Hard%', 'EV',␣
 ↪'LA', 'Barrel%', 'HardHit%', \
                                           'O-Swing%', 'Z-Swing%', 'Swing%',␣
 ↪'O-Contact%', 'Z-Contact%', \
                                           'Contact%', 'Zone%', 'F-Strike%',␣
 ↪'SwStr%', 'CStr%', 'CSW%', 'W', 'L'])

# Removing the % in the values so that they can be used as numbers
team_batting['BB%'] = team_batting['BB%'].replace({'\%':''}, regex = True)
team_batting['K%'] = team_batting['K%'].replace({'\%':''}, regex = True)
team_batting['LD%'] = team_batting['BB%'].replace({'\%':''}, regex = True)
team_batting['GB%'] = team_batting['GB%'].replace({'\%':''}, regex = True)
team_batting['FB%'] = team_batting['FB%'].replace({'\%':''}, regex = True)
team_batting['HR/FB'] = team_batting['HR/FB'].replace({'\%':''}, regex = True)
team_batting['Pull%'] = team_batting['Pull%'].replace({'\%':''}, regex = True)
```

```python
team_batting['Cent%'] = team_batting['Cent%'].replace({'\%':''}, regex = True)
team_batting['Oppo%'] = team_batting['Oppo%'].replace({'\%':''}, regex = True)
team_batting['Soft%'] = team_batting['Soft%'].replace({'\%':''}, regex = True)
team_batting['Med%'] = team_batting['Med%'].replace({'\%':''}, regex = True)
team_batting['Hard%'] = team_batting['Hard%'].replace({'\%':''}, regex = True)
team_batting['Barrel%'] = team_batting['Barrel%'].replace({'\%':''}, regex = True)
team_batting['HardHit%'] = team_batting['HardHit%'].replace({'\%':''}, regex = True)
team_batting['O-Swing%'] = team_batting['O-Swing%'].replace({'\%':''}, regex = True)
team_batting['Z-Swing%'] = team_batting['Z-Swing%'].replace({'\%':''}, regex = True)
team_batting['Swing%'] = team_batting['Swing%'].replace({'\%':''}, regex = True)
team_batting['O-Contact%'] = team_batting['O-Contact%'].replace({'\%':''}, regex = True)
team_batting['Z-Contact%'] = team_batting['Z-Contact%'].replace({'\%':''}, regex = True)
team_batting['Contact%'] = team_batting['Contact%'].replace({'\%':''}, regex = True)
team_batting['Zone%'] = team_batting['Zone%'].replace({'\%':''}, regex = True)
team_batting['F-Strike%'] = team_batting['F-Strike%'].replace({'\%':''}, regex = True)
team_batting['SwStr%'] = team_batting['SwStr%'].replace({'\%':''}, regex = True)
team_batting['CStr%'] = team_batting['CStr%'].replace({'\%':''}, regex = True)
team_batting['CSW%'] = team_batting['CSW%'].replace({'\%':''}, regex = True)



# Making all values numeric if they have only numbers
team_batting = team_batting.apply(pd.to_numeric, errors='ignore')

# Replace zero values with NaN (because some years don't have data for certain newer stats
team_batting['EV'] = team_batting['EV'].replace(0.0, np.nan)
team_batting['LA'] = team_batting['LA'].replace(0.0, np.nan)
team_batting['Barrel%'] = team_batting['Barrel%'].replace(0.0, np.nan)
team_batting['HardHit%'] = team_batting['HardHit%'].replace(0.0, np.nan)

# Reordering columns
team_batting = team_batting[['Year', 'Team', 'W', 'L', 'AB', 'PA', 'AVG', \
                             'H', '1B', '2B', '3B', 'HR', 'R', 'RBI', \
                             'BB', 'IBB', 'SO', 'HBP', 'SF', 'SH', 'GDP', \
                             'SB', 'CS', 'BB%', 'K%', 'BB/K', 'OBP', 'SLG', \
                             'OPS', 'ISO', 'BABIP', 'wOBA', 'wRC+', 'GB/FB', \
```

```
                                     'LD%', 'GB%', 'FB%', 'HR/FB', 'EV', 'LA',␣
↪'Barrel%', \
                                     'HardHit%', 'O-Swing%', 'Z-Swing%', 'Swing%',␣
↪'O-Contact%', 'Z-Contact%', 'Contact%']]
team_batting
```

[4]:

```
     Year Team    W    L    AB    PA    AVG     H    1B   2B  3B   HR    R  \
0    2002  ANA   99   63  5678  6327  0.282  1603  1086  333  32  152  851
1    2002  ARI   98   64  5508  6318  0.267  1471   982  283  41  165  819
2    2002  ATL  101   59  5495  6224  0.260  1428   959  280  25  164  708
3    2002  BAL   67   95  5491  6096  0.246  1353   850  311  27  165  667
4    2002  BOS   93   69  5640  6332  0.277  1560  1002  348  33  177  859
..    …    …    …    …    …     …     …    …    …    …  ..   …    …
541  2021  SFG  107   55  5462  6196  0.249  1360   823  271  25  241  804
542  2021  STL   90   72  5351  6001  0.244  1303   822  261  22  198  706
543  2021  TEX   60  102  5405  5943  0.232  1254   838  225  24  167  625
544  2021  TOR   91   71  5476  6070  0.266  1455   895  285  13  262  846
545  2021  WSN   65   97  5385  6113  0.258  1388   914  272  20  182  724

     RBI   BB  IBB    SO  HBP  SF  SH  GDP   SB  CS   BB%    K%  BB/K    OBP  \
0    811  462   42   805   74  64  49  105  117  51   7.3  12.7  0.57  0.341
1    783  643   58  1016   50  53  62  130   92  46  10.2  16.1  0.63  0.346
2    669  558   68  1028   54  49  67  147   76  39   9.0  16.5  0.54  0.331
3    636  452   25   993   64  49  40  128  110  48   7.4  16.3  0.46  0.309
4    810  545   39   944   72  53  22  139   80  28   8.6  14.9  0.58  0.345
..    …    …    …    …    …  ..  ..    …    …  ..     …     …     …      …
541  768  602   45  1461   64  30  36  117   66  14   9.7  23.6  0.41  0.329
542  678  478   32  1341   86  44  40   99   89  22   8.0  22.3  0.36  0.313
543  598  433   10  1381   58  31  16  113  106  29   7.3  23.2  0.31  0.294
544  816  496   14  1218   51  35  10  112   81  20   8.2  20.1  0.41  0.330
545  686  573   43  1303   84  31  38  158   56  26   9.4  21.3  0.44  0.337

       SLG    OPS    ISO  BABIP   wOBA  wRC+  GB/FB   LD%   GB%   FB%  HR/FB  \
0    0.433  0.773  0.150  0.303  0.336   105   1.04   7.3  39.4  38.1    8.2
1    0.423  0.769  0.156  0.298  0.335    97   1.36  10.2  45.2  33.2   11.1
2    0.409  0.741  0.150  0.290  0.322    94   1.39   9.0  46.3  33.2   11.1
3    0.403  0.712  0.157  0.271  0.311    90   1.05   7.4  41.0  39.2    9.4
4    0.444  0.789  0.168  0.302  0.343   107   1.22   8.6  43.1  35.4   10.6
..      …      …      …      …      …     …      …     …     …     …      …
541  0.440  0.769  0.191  0.295  0.329   108   1.03   9.7  39.7  38.5   15.6
542  0.412  0.725  0.168  0.287  0.312    97   1.04   8.0  40.5  38.9   12.6
543  0.375  0.670  0.143  0.280  0.291    84   1.34   7.3  46.4  34.6   12.0
544  0.466  0.797  0.200  0.296  0.340   112   1.04   8.2  40.4  38.8   15.8
545  0.417  0.754  0.159  0.307  0.326   101   1.54   9.4  47.4  30.8   14.5

       EV    LA  Barrel%  HardHit%  O-Swing%  Z-Swing%  Swing%  O-Contact%  \
0     NaN   NaN      NaN       NaN      18.1      69.9    47.1        55.1
```

```
1      NaN   NaN       NaN        NaN       15.6       67.6    43.7            46.6
2      NaN   NaN       NaN        NaN       17.1       72.4    47.4            47.7
3      NaN   NaN       NaN        NaN       18.9       70.8    47.3            52.5
4      NaN   NaN       NaN        NaN       17.8       70.7    46.4            49.0
..     …    …       …       …       …       …      …         …
541  88.7  14.8       9.3      38.7       28.2       68.8    45.6            62.5
542  88.7  14.0       7.9      37.2       31.5       68.2    47.0            64.2
543  87.9  10.9       6.6      35.6       32.6       68.8    47.9            64.1
544  90.3  13.6       9.7      42.2       31.9       72.5    48.8            63.3
545  88.6   9.5       6.9      39.2       29.2       68.0    45.7            62.7

     Z-Contact%  Contact%
0          88.3      82.7
1          86.3      79.8
2          84.6      78.6
3          87.1      80.8
4          87.6      80.8
..          …        …
541        84.6      76.7
542        84.4      76.6
543        83.9      76.1
544        86.7      77.8
545        86.0      77.4

[546 rows x 48 columns]
```

Correlation Between Scoring Runs and Various Batting Metrics Since the team that has more runs wins the game, run are directly correlated to winning games. Obviously, that is a generic statement that can have some nuance; of course, a team that scores a lot of runs but gives up even more runs, will lose games, so really a team's Run%, Runs Scored / (Runs Scored + Runs Scored), is more directly related to winning, but we aren't worried about defense for this exercise. Below, we are going to try to find the offensive metric(s) that best correlate with scoring runs, because scoring runs wins games, to an extent. We will plot the important metrics, described below, against a team's run total and find the correlation between the datapoints. This will show which stat is most correlated to scoring runs, and thus the stat that is likely important in terms of helping a team win games. Below are the metrics that we will be analyzing:

AVG: Batting Average > The percentage of times the batter gets a hit of out of all of his at-bats. (H/AB) **Formula:** H / AB

OBP: On-Base Percentage > The ratio of the sum of the batter's hits, walks, hit by pitches to their number of plate appearances. **Formula:** (H + BB + IBB + HBP) / PA

SLG: Slugging Percentage > The total number of bases a player records per at-bat **Formula:** (1B + 2(2B) + 3(3B) + HR)/AB

OPS: On-Base Plus Slugging Percentage > Measures the ability of a player both to get on base and to hit for power **Formula:** OBP + SLG

wOBA: Weighted On-Base Average > Designed to measure a player's overall offensive contributions per plate appearance **Formula:** (0.69 * NIBB) + (0.719 * HBP) + (0.87 * 1B) + (1.217 * 2B) + (1.529 * 3B) + (1.94 * HR) / (AB + BB - IBB + SF + HBP)

SLOB: Slugging Times On-Base > **Formula:** SLG * OBP

[5]:
```
# 2020 was shortened due to COVID, so only 60 regular season games were played␣
↪meaning less runs were scored,
# so we will ignore that for this exercise
team = team_batting[team_batting.Year != 2020]

fig, ax = plt.subplots(3, 3)
fig.subplots_adjust(wspace=.25)
fig.set_figheight(25)
fig.set_figwidth(35)
fig.suptitle("Correlation Between Runs Scored and Various Batting Metrics",␣
↪fontsize=40)


######################################################################
#       Plotting Correlation Batting Average vs. Runs Scored         #
######################################################################

plt.sca(ax[0,0])
plt.gca().set_title('Batting Average vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Batting Average (AVG)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['AVG'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['AVG'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
↪color='red')
plt.scatter(team['AVG'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#          Plotting Correlation Home Runs vs. Runs Scored            #
```

```
######################################################################

plt.sca(ax[0,1])
plt.gca().set_title('Home Runs vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Home Runs (HR)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['HR'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['HR'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
 ↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
 ↪color='red')
plt.scatter(team['HR'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#            Plotting Correlation OBP vs. Runs Scored                #
######################################################################

plt.sca(ax[0,2])
plt.gca().set_title('On-Base Percentage vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('On-Base Percentage (OBP)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['OBP'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['OBP'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
 ↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
```

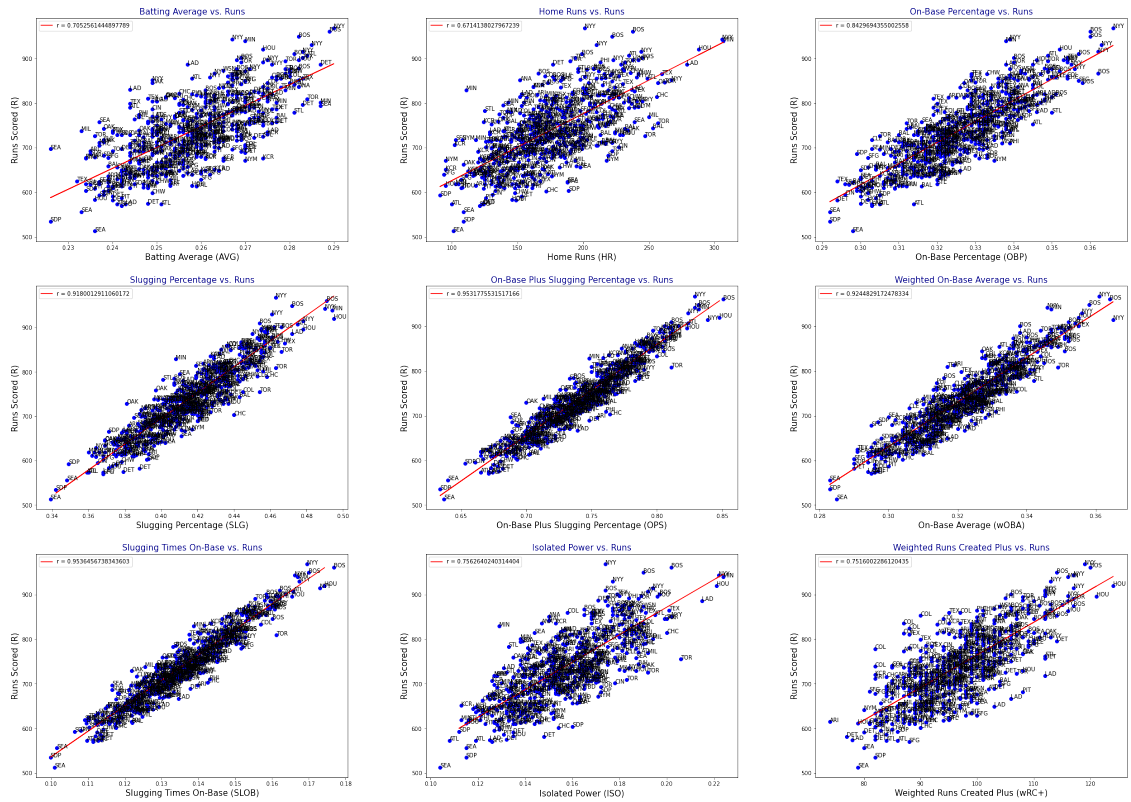
```
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
↪color='red')
plt.scatter(team['OBP'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#           Plotting Correlation SLG vs. Runs Scored                 #
######################################################################

plt.sca(ax[1,0])
plt.gca().set_title('Slugging Percentage vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Slugging Percentage (SLG)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['SLG'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['SLG'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
↪color='red')
plt.scatter(team['SLG'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#           Plotting Correlation OPS vs. Runs Scored                 #
######################################################################

plt.sca(ax[1,1])
plt.gca().set_title('On-Base Plus Slugging Percentage vs. Runs', fontsize=15, c␣
↪= 'DarkBlue')
plt.gca().set_xlabel('On-Base Plus Slugging Percentage (OPS)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['OPS'], row['R']))

# Calculate regression line and plot it in the same graph
```

```python
model = LinearRegression()
x, y = team['OPS'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}', color='red')
plt.scatter(team['OPS'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#           Plotting Correlation wOBA vs. Runs Scored                #
######################################################################

plt.sca(ax[1,2])
plt.gca().set_title('Weighted On-Base Average vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('On-Base Average (wOBA)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['wOBA'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['wOBA'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}', color='red')
plt.scatter(team['wOBA'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#            Plotting Correlation SLOB vs. Runs Scored               #
######################################################################

plt.sca(ax[2,0])
```

```python
plt.gca().set_title('Slugging Times On-Base vs. Runs', fontsize=15, c =
 'DarkBlue')
plt.gca().set_xlabel('Slugging Times On-Base (SLOB)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['SLOB'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['SLOB'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,
 train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',
 color='red')
plt.scatter(team['SLOB'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#            Plotting Correlation SLOB vs. Runs Scored               #
######################################################################

plt.sca(ax[2,1])
plt.gca().set_title('Isolated Power vs. Runs', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Isolated Power (ISO)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['ISO'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['ISO'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,
 train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
```

```
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
 ↪color='red')
plt.scatter(team['ISO'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#            Plotting Correlation wRC+ vs. Runs Scored               #
######################################################################

plt.sca(ax[2,2])
plt.gca().set_title('Weighted Runs Created Plus vs. Runs', fontsize=15, c =␣
 ↪'DarkBlue')
plt.gca().set_xlabel('Weighted Runs Created Plus (wRC+)', fontsize=15)
plt.gca().set_ylabel('Runs Scored (R)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['wRC+'], row['R']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['wRC+'].values.reshape(-1,1), team['R'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
 ↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
 ↪color='red')
plt.scatter(team['wRC+'], team['R'], color = 'blue')
plt.legend(loc = 'upper left')
plt.show()
```

Correlation Between Runs Scored and Various Batting Metrics



A baseball fan with basic knowledge might be under the assumption that a batting average can determine whether or not a player is good at hitting. As the plots have shown, this is not exactly the case. At the end of the day, teams want to score runs, regardless of how they do so. The plots, however, show that out of the five metrics we studied, batting average was the least correlated to scoring runs, with a correlation coefficient of just .705. The metric that had the greatest correlation to scoring runs was Slugging Times On-Base (SLOB), with a marginally close second in On-Base Plus Slugging (OPS), both with a correlaton coefficient of .953.

What we can gather from this is that teams should value a player with a high SLOB and high OPS rather than just a looking at AVG and HR like we used to. A player who has a batting average of .330 but only hits singles and hardly ever walks is going to be less valueable than a player who hits .330 but all of his hits are extra base hits on top of working walks.

Correlation Between Plate Discipline and Scoring Runs Now, with SLOB, we have an offensive metric that we determined to be highly correlated to scoring runs. Next, we want to determine what metrics are going to correlate to having a high SLOB rating. One of the most important skills a player can have, that diligent teams stress on, is plate discipline. In an era where strikeouts are happening at historic rates, having a player with a keen batting eye can be the difference between starting a rally and ending one. The metrics we will look at are below:

BB/K: Walk to Strikeout Rate Rate > The rate at which a batter walks compared to stirking out. A value over 1 means that the batter walks more than he strikes out and a value under 1 means that he strikes out more than he walks.

O-Swing%: Swing Rate on Pitches Outside the Strike Zone > The percentage of pitches that are outside of the strike zone that the batter swings at.

Z-Swing%: Swing Rate on Pitches Inside the Strike Zone > The percentage of pitches that are inside of the strike zone that the batter swings at.

Swing%: Swing Rate > The percentage of pitches that the batter swings at.

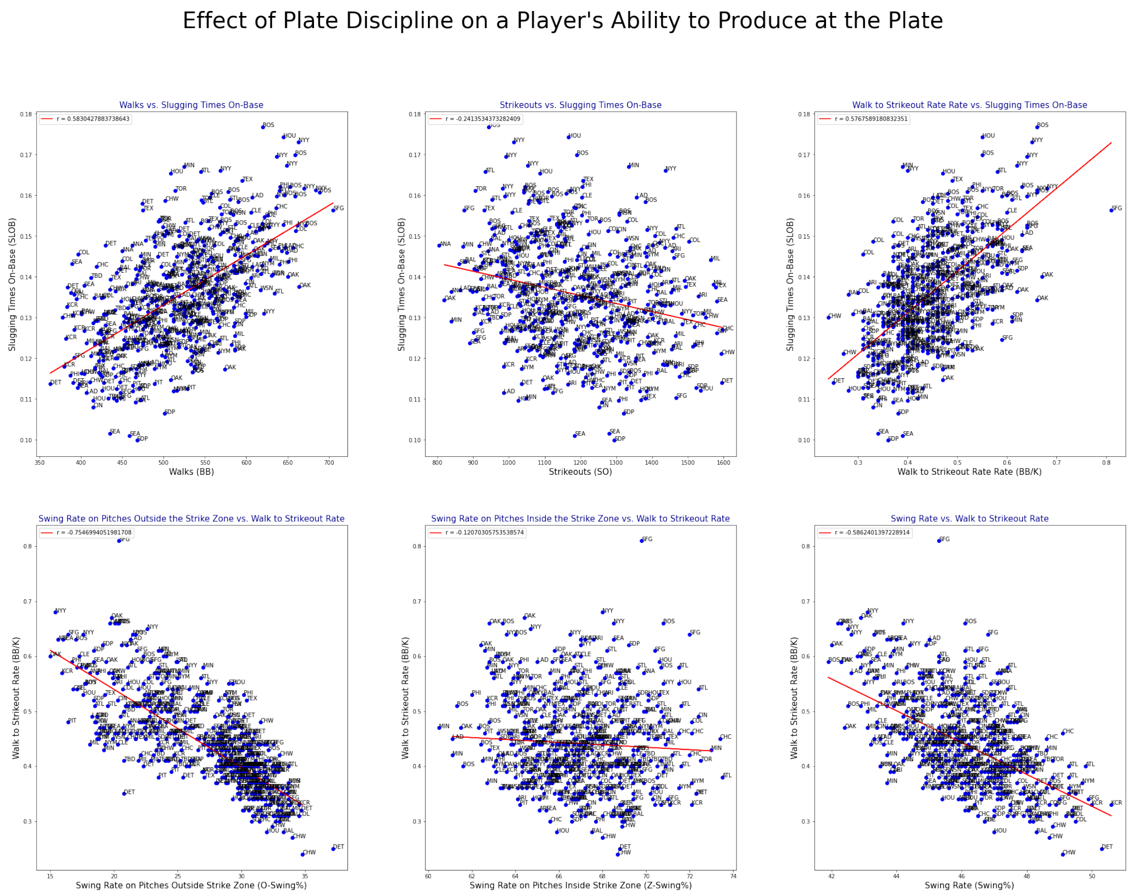
[6]:
```
team = team_batting[team_batting.Year != 2020]

fig, ax = plt.subplots(2, 3)
fig.subplots_adjust(wspace=.25)
fig.set_figheight(25)
fig.set_figwidth(35)
fig.suptitle("Effect of Plate Discipline on a Player's Ability to Produce at␣
↪the Plate", fontsize=40)


######################################################################
#             Plotting Correlation Walks vs. SLOB                    #
######################################################################

plt.sca(ax[0,0])
plt.gca().set_title('Walks vs. Slugging Times On-Base', fontsize=15, c =␣
↪'DarkBlue')
plt.gca().set_xlabel('Walks (BB)', fontsize=15)
plt.gca().set_ylabel('Slugging Times On-Base (SLOB)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['BB'], row['SLOB']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['BB'].values.reshape(-1,1), team['SLOB'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
↪color='red')
plt.scatter(team['BB'], team['SLOB'], color = 'blue')
plt.legend(loc = 'upper left')
```

```
######################################################################
#           Plotting Correlation Strikeouts vs. SLOB                 #
######################################################################

plt.sca(ax[0,1])
plt.gca().set_title('Strikeouts vs. Slugging Times On-Base', fontsize=15, c =␣
 ↪'DarkBlue')
plt.gca().set_xlabel('Strikeouts (SO)', fontsize=15)
plt.gca().set_ylabel('Slugging Times On-Base (SLOB)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['SO'], row['SLOB']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['SO'].values.reshape(-1,1), team['SLOB'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
 ↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
 ↪color='red')
plt.scatter(team['SO'], team['SLOB'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#               Plotting Correlation BB/K vs. SLOB.                  #
######################################################################

plt.sca(ax[0,2])
plt.gca().set_title('Walk to Strikeout Rate Rate vs. Slugging Times On-Base',␣
 ↪fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Walk to Strikeout Rate Rate (BB/K)', fontsize=15)
plt.gca().set_ylabel('Slugging Times On-Base (SLOB)', fontsize=15)

team = team.assign(SLOB = team.SLG * team.OBP)
# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['BB/K'], row['SLOB']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
```

```
x, y = team['BB/K'].values.reshape(-1,1), team['SLOB'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}', color='red')
plt.scatter(team['BB/K'], team['SLOB'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#                Plotting Correlation BB/K vs. SLOB.                 #
######################################################################

plt.sca(ax[1,0])
plt.gca().set_title('Swing Rate on Pitches Outside the Strike Zone vs. Walk to Strikeout Rate', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Swing Rate on Pitches Outside Strike Zone (O-Swing%)', fontsize=15)
plt.gca().set_ylabel('Walk to Strikeout Rate (BB/K)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['O-Swing%'], row['BB/K']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['O-Swing%'].values.reshape(-1,1), team['BB/K'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}', color='red')
plt.scatter(team['O-Swing%'], team['BB/K'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#                Plotting Correlation BB/K vs. SLOB.                 #
######################################################################

plt.sca(ax[1,1])
```

```
plt.gca().set_title('Swing Rate on Pitches Inside the Strike Zone vs. Walk to␣
↪Strikeout Rate', fontsize=15, c = 'DarkBlue')
plt.gca().set_xlabel('Swing Rate on Pitches Inside Strike Zone (Z-Swing%)',␣
↪fontsize=15)
plt.gca().set_ylabel('Walk to Strikeout Rate (BB/K)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['Z-Swing%'], row['BB/K']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['Z-Swing%'].values.reshape(-1,1), team['BB/K'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)

# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',␣
↪color='red')
plt.scatter(team['Z-Swing%'], team['BB/K'], color = 'blue')
plt.legend(loc = 'upper left')

######################################################################
#              Plotting Correlation BB/K vs. SLOB.                   #
######################################################################

plt.sca(ax[1,2])
plt.gca().set_title('Swing Rate vs. Walk to Strikeout Rate', fontsize=15, c =␣
↪'DarkBlue')
plt.gca().set_xlabel('Swing Rate (Swing%)', fontsize=15)
plt.gca().set_ylabel('Walk to Strikeout Rate (BB/K)', fontsize=15)

# Add labels for each point
for idx, row in team.iterrows():
    plt.annotate(row['Team'], (row['Swing%'], row['BB/K']))

# Calculate regression line and plot it in the same graph
model = LinearRegression()
x, y = team['Swing%'].values.reshape(-1,1), team['BB/K'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.3)
model = model.fit(x_train, y_train)
prediction = model.predict(x_test)
```

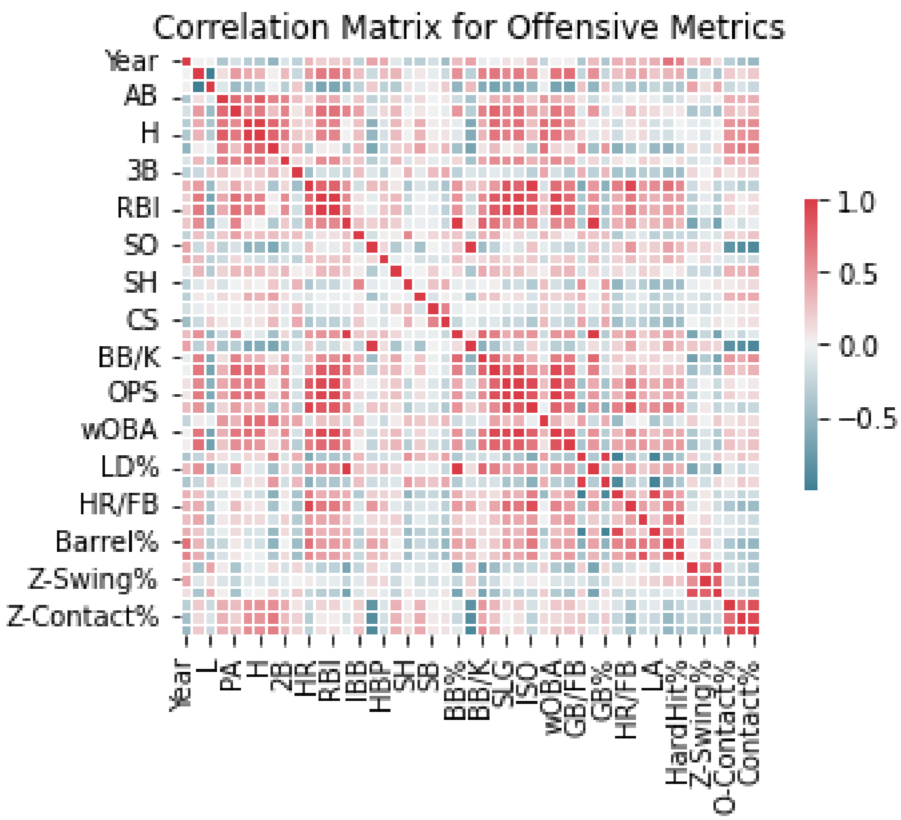
```python
# Plot regression line and scatter the data points on the same axis
plt.plot(x_test, prediction, label = f'r = {pearsonr(x.flatten(), y)[0]}',
 ↪color='red')
plt.scatter(team['Swing%'], team['BB/K'], color = 'blue')
plt.legend(loc = 'upper left')

plt.show()
```



As the plots show, a batter's strikeout to walk rate is moderately correlated to a player's ability to produce at the plate, as it has a .577 correlation coefficient. Furthermore, after looking at how plate disicpline effects a batter's strikeout to walk rate, we determined that the correlation between a batter having a low BB/K and a batter swinging at pitches outside of the strike zone is strong. In addition, we also found that the more pitches that a batter swings overall will lead to a decrease in BB/K. Through this, we can conclude that in order for a batter to be productive at the plate, it's important for them to make smart swing decisions, meaning that they should be selective of what pitches to swing at; minimizing the number of pitches that are outside of the strike zone that a batter swings at will be veryu beneficial to improving their BB/K and consequently improving their overall production with the bat in their hands.

Correlation Matrix for Offensive Metrics

```python
[7]: team = team_batting[team_batting.Year != 2020]
     team = team[team.Year > 2014]

     corr = team.corr()
     mask = np.triu(np.ones_like(corr, dtype=bool))
     cmap = seaborn.diverging_palette(220, 10, as_cmap=True)
     seaborn.heatmap(corr, cmap=cmap, vmax=1, center=0, square=True, linewidths=.5, cbar_kws={"shrink": .5})
     plt.title('Correlation Matrix for Offensive Metrics')
     plt.show()
```



0.0.3 Part II: Scraping Player Data for 2011-2021 Seasons

Scraping Standard Player Data From Fangraphs

```python
[8]: pages = [i for i in range(1, 14)]
     team_idx = [i for i in range(1, 31)]
     teams = ['LAA', 'BAL', 'BOS', 'CHW', 'CLE', 'DET', 'KCR', 'MIN', 'NYY', 'OAK', \
              'SEA', 'TBR', 'TEX', 'TOR', 'ARI', 'ATL', 'CHC', 'CIN', 'COL', 'MIA', \
              'HOU', 'LAD', 'MIL', 'WSN', 'NYM', 'PHI', 'PIT', 'STL', 'SDP', 'SFG']
     urls = []
```

```python
for team in team_idx:
    for page in pages:
        urls.append((get_urls(team, (2010, 2021), page, 'player_standard'),␣
 ↪teams[team-1]))

count = 0
for url in urls:
    if count == 0:
        player_standard = scraping_FanGraphs(url[0], None, url[1])
        count = 1
    else:
        player_standard = pd.concat([player_standard,␣
 ↪scraping_FanGraphs(url[0], None, url[1])])
player_standard
```

[8]:

```
      # Season            Name   G  AB  PA  H  1B  2B  3B  HR  R  RBI  BB  IBB  SO  HBP  \
0     1   2018   Juan Graterol   1   1   1  1   1   0   0   0  0    0   0    0   0    0
1     2   2011  Tyler Chatwood  27   3   5  2   2   0   0   0  1    0   0    0   0    0
2     3   2011   Gil Velazquez   4   6   7  3   3   0   0   0  0    1   0    0   0    0
3     4   2011   Ervin Santana  33   2   2  1   1   0   0   0  0    0   0    0   1    0
4     5   2012    Jered Weaver  30   2   3  1   1   0   0   0  1    0   1    0   1    0
..   …      …              …  ..  ..  .. ..  ..  ..  ..  .. ..   ..  ..   ..  ..   ..
19  570   2021    Conner Menez   8   0   0  0   0   0   0   0  0    0   0    0   0    0
20  571   2021   Caleb Baragar  25   2   2  0   0   0   0   0  0    0   0    0   2    0
21  572   2021    Kervin Castro  10   0   0  0   0   0   0   0  0    0   0    0   0    0
22  573   2021  Gregory Santos   3   0   0  0   0   0   0   0  0    0   0    0   0    0
23  574   2021    Camilo Doval  29   0   0  0   0   0   0   0  0    0   0    0   0    0

   SF  SH  GDP  SB  CS    AVG  Year  Team
0   0   0    0   0   0  1.000  None   LAA
1   0   2    0   0   0   .667  None   LAA
2   1   0    0   0   0   .500  None   LAA
3   0   0    0   0   0   .500  None   LAA
4   0   0    0   0   0   .500  None   LAA
..  ..  ..  ..  ..  ..     …     …     …
19  0   0    0   0   0   .000  None   SFG
20  0   0    0   0   0   .000  None   SFG
21  0   0    0   0   0   .000  None   SFG
22  0   0    0   0   0   .000  None   SFG
23  0   0    0   0   0   .000  None   SFG

[18060 rows x 25 columns]
```

Scraping Advanced Player Data From Fangraphs

[9]:
```python
pages = [i for i in range(1, 14)]
team_idx = [i for i in range(1, 31)]
teams = ['LAA', 'BAL', 'BOS', 'CHW', 'CLE', 'DET', 'KCR', 'MIN', 'NYY', 'OAK', \
         'SEA', 'TBR', 'TEX', 'TOR', 'ARI', 'ATL', 'CHC', 'CIN', 'COL', 'MIA', \
         'HOU', 'LAD', 'MIL', 'WSN', 'NYM', 'PHI', 'PIT', 'STL', 'SDP', 'SFG']
urls = []

for team in team_idx:
    for page in pages:
        urls.append((get_urls(team, (2010, 2021), page, 'player_advanced'),␣
 ↪teams[team-1]))

count = 0
for url in urls:
    if count == 0:
        player_advanced = scraping_FanGraphs(url[0], None, url[1])
        count = 1
    else:
        player_advanced = pd.concat([player_advanced,␣
 ↪scraping_FanGraphs(url[0], None, url[1])])
player_advanced
```

[9]:
```
      #  Season            Name  PA     BB%     K%  BB/K    AVG    OBP    SLG  \
0     1    2015      Jett Bandy   2    0.0%    0.0%  0.00   .500   .500  2.000
1     2    2018   Juan Graterol   1    0.0%    0.0%  0.00  1.000  1.000  1.000
2     3    2013     John Hester   1  100.0%    0.0%  1.00   .000  1.000   .000
3     4    2011  Tyler Chatwood   5    0.0%    0.0%  0.00   .667   .667   .667
4     5    2010     Ryan Budde  11    9.1%   45.5%  0.20   .400   .455   .800
..  …     …               …  ..   …      …   …    …      …      …
19  570    2021    Conner Menez   0    0.0%    0.0%  0.00   .000   .000   .000
20  571    2021   Caleb Baragar   2    0.0%  100.0%  0.00   .000   .000   .000
21  572    2021   Kervin Castro   0    0.0%    0.0%  0.00   .000   .000   .000
22  573    2021  Gregory Santos   0    0.0%    0.0%  0.00   .000   .000   .000
23  574    2021    Camilo Doval   0    0.0%    0.0%  0.00   .000   .000   .000

      OPS    ISO  Spd  BABIP   UBR wGDP  wSB wRC  wRAA   wOBA  wRC+  Year Team
0   2.500  1.500  0.1   .000   0.0  0.0  0.0   1   1.1  1.033   597  None  LAA
1   2.000   .000  0.1  1.000   0.0  0.0  0.0   1   0.5   .880   484  None  LAA
2   1.000   .000  2.6   .000   0.0  0.0  0.0   0   0.3   .690   361  None  LAA
3   1.333   .000  2.6   .667   0.0  0.0  0.0   2   1.1   .594   289  None  LAA
4   1.255   .400  1.1   .750  -0.1  0.0  0.0   3   1.8   .530   244  None  LAA
..    …     …  …    …     …  …  …  ..   …     …   …   …  …
19   .000   .000  0.1   .000   0.0  0.0  0.0   0   0.0   .000   0.0  None  SFG
20   .000   .000  0.1   .000   0.0  0.0  0.0   0  -0.5   .000  -100  None  SFG
21   .000   .000  0.1   .000   0.0  0.0  0.0   0   0.0   .000   0.0  None  SFG
22   .000   .000  0.1   .000   0.0  0.0  0.0   0   0.0   .000   0.0  None  SFG
23   .000   .000  0.1   .000   0.0  0.0  0.0   0   0.0   .000   0.0  None  SFG
```

```
[18060 rows x 23 columns]
```

Scraping Batted Ball Player Data From Fangraphs

[10]:
```python
pages = [i for i in range(1, 14)]
team_idx = [i for i in range(1, 31)]
teams = ['LAA', 'BAL', 'BOS', 'CHW', 'CLE', 'DET', 'KCR', 'MIN', 'NYY', 'OAK', \
         'SEA', 'TBR', 'TEX', 'TOR', 'ARI', 'ATL', 'CHC', 'CIN', 'COL', 'MIA', \
         'HOU', 'LAD', 'MIL', 'WSN', 'NYM', 'PHI', 'PIT', 'STL', 'SDP', 'SFG']
urls = []

for team in team_idx:
    for page in pages:
        urls.append((get_urls(team, (2010, 2021), page, 'player_batted'), teams[team-1]))

count = 0
for url in urls:
    if count == 0:
        player_batted = scraping_FanGraphs(url[0], None, url[1])
        count = 1
    else:
        player_batted = pd.concat([player_batted, scraping_FanGraphs(url[0], None, url[1])])
player_batted
```

[10]:
```
      #  Season             Name  BABIP  GB/FB    LD%    GB%     FB%  IFFB%  \
0     1    2018     Ryan Schimpf   .000   1.00   0.0%  50.0%   50.0%   0.0%
1     2    2019     Cesar Puello   .433   4.40  18.2%  66.7%   15.2%   0.0%
2     3    2010       Ryan Budde   .750   0.00  60.0%   0.0%   40.0%   0.0%
3     4    2015       Jett Bandy   .000   0.00   0.0%   0.0%  100.0%   0.0%
4     5    2018    Nolan Fontana   .000   0.33  20.0%  20.0%   60.0%   0.0%
..   …      …              …      …      …      …      …       …      …
19   570   2021        Joey Bart   .500   1.00  50.0%  25.0%   25.0%   0.0%
20   571   2021    Kervin Castro   .000   0.00   0.0%   0.0%    0.0%   0.0%
21   572   2021   Gregory Santos   .000   0.00   0.0%   0.0%    0.0%   0.0%
22   573   2021     Camilo Doval   .000   0.00   0.0%   0.0%    0.0%   0.0%
23   574   2021       Sammy Long   .167   4.00  33.3%  66.7%    0.0%   0.0%

     HR/FB IFH    IFH% BUH   BUH%  Pull%  Cent%  Oppo%  Soft%   Med%  Hard%  \
0   100.0%   0    0.0%   0   0.0%   0.0%  50.0%  50.0%  50.0%   0.0%  50.0%
1    60.0%   3   13.6%   0   0.0%  48.5%  33.3%  18.2%  24.2%  36.4%  39.4%
2    50.0%   0    0.0%   0   0.0%  40.0%  40.0%  20.0%   0.0%  80.0%  20.0%
3    50.0%   0    0.0%   0   0.0%  50.0%   0.0%  50.0%   0.0%  50.0%  50.0%
4    33.3%   0    0.0%   0   0.0%  60.0%  20.0%  20.0%   0.0%  60.0%  40.0%
..     …   ..     …   ..     …      …      …      …      …      …      …
```

```
19    0.0%   1  100.0%   0  0.0%    0.0%  75.0%  25.0%  50.0%  50.0%   0.0%
20    0.0%   0    0.0%   0  0.0%    0.0    0.0    0.0    0.0    0.0     0.0
21    0.0%   0    0.0%   0  0.0%    0.0    0.0    0.0    0.0    0.0     0.0
22    0.0%   0    0.0%   0  0.0%    0.0    0.0    0.0    0.0    0.0     0.0
23    0.0%   0    0.0%   0  0.0%  14.3%  42.9%  42.9%  42.9%  42.9%  14.3%

    Year Team
0   None  LAA
1   None  LAA
2   None  LAA
3   None  LAA
4   None  LAA
..   …   …
19  None  SFG
20  None  SFG
21  None  SFG
22  None  SFG
23  None  SFG

[18060 rows x 22 columns]
```

Scraping Statcast Player Data From Fangraphs

[11]:
```
pages = [i for i in range(1, 14)]
team_idx = [i for i in range(1, 31)]
teams = ['LAA', 'BAL', 'BOS', 'CHW', 'CLE', 'DET', 'KCR', 'MIN', 'NYY', 'OAK', \
         'SEA', 'TBR', 'TEX', 'TOR', 'ARI', 'ATL', 'CHC', 'CIN', 'COL', 'MIA', \
         'HOU', 'LAD', 'MIL', 'WSN', 'NYM', 'PHI', 'PIT', 'STL', 'SDP', 'SFG']
urls = []

for team in team_idx:
    for page in pages:
        urls.append((get_urls(team, (2010, 2021), page, 'player_statcast'),␣
 ↪teams[team-1]))

count = 0
for url in urls:
    if count == 0:
        player_statcast = scraping_FanGraphs(url[0], None, url[1])
        count = 1
    else:
        player_statcast = pd.concat([player_statcast,␣
 ↪scraping_FanGraphs(url[0], None, url[1])])
player_statcast
```

[11]:
```
      # Season              Name  PA Events    EV  maxEV     LA Barrels  \
0     1   2020  Franklin Barreto  18       9  95.6  109.2   10.4        0
```

```
1     2   2019       Tyler Skaggs   3      1  95.6   95.6  -36.0        0
2     3   2018       Jabari Blash  45     16  94.6  116.2   17.9        2
3     4   2021     Andrew Heaney   3      1  94.1   94.1   10.2        0
4     5   2018         Joe Hudson  12     12  94.0  103.9   12.8        0
..   …    …                 …  ..   …    …    …     …      …
19   570  2021      Conner Menez   0      0   0.0    0.0    0.0      0.0
20   571  2021     Caleb Baragar   2      0   0.0    0.0    0.0        0
21   572  2021     Kervin Castro   0      0   0.0    0.0    0.0      0.0
22   573  2021    Gregory Santos   0      0   0.0    0.0    0.0      0.0
23   574  2021      Camilo Doval   0      0   0.0    0.0    0.0      0.0

   Barrel% HardHit HardHit%   AVG  xBA   SLG xSLG  wOBA xwOBA  Year Team
0     0.0%       5    55.6%  .118  0.0  .118  0.0  .139   0.0  None  LAA
1     0.0%       1   100.0%  .000  0.0  .000  0.0  .230   0.0  None  LAA
2    12.5%       8    50.0%  .103  0.0  .128  0.0  .163   0.0  None  LAA
3     0.0%       0     0.0%  .500  0.0  .500  0.0  .524   0.0  None  LAA
4     0.0%       7    58.3%  .167  0.0  .250  0.0  .177   0.0  None  LAA
..     …      …       …    …   …    …   …    …    …    …   …
19     0.0     0.0      0.0  .000  0.0  .000  0.0  .000   0.0  None  SFG
20     0.0       0      0.0  .000  0.0  .000  0.0  .000   0.0  None  SFG
21     0.0     0.0      0.0  .000  0.0  .000  0.0  .000   0.0  None  SFG
22     0.0     0.0      0.0  .000  0.0  .000  0.0  .000   0.0  None  SFG
23     0.0     0.0      0.0  .000  0.0  .000  0.0  .000   0.0  None  SFG

[18060 rows x 20 columns]
```

Scraping Plate Discipline Player Data From Fangraphs

[12]:
```python
pages = [i for i in range(1, 14)]
team_idx = [i for i in range(1, 31)]
teams = ['LAA', 'BAL', 'BOS', 'CHW', 'CLE', 'DET', 'KCR', 'MIN', 'NYY', 'OAK', \
         'SEA', 'TBR', 'TEX', 'TOR', 'ARI', 'ATL', 'CHC', 'CIN', 'COL', 'MIA', \
         'HOU', 'LAD', 'MIL', 'WSN', 'NYM', 'PHI', 'PIT', 'STL', 'SDP', 'SFG']
urls = []

for team in team_idx:
    for page in pages:
        urls.append((get_urls(team, (2010, 2021), page,␣
 ↪'player_plate_discipline'), teams[team-1]))

count = 0
for url in urls:
    if count == 0:
        player_plate_discipline = scraping_FanGraphs(url[0], None, url[1])
        count = 1
    else:
```

```
    player_plate_discipline = pd.concat([player_plate_discipline,␣
↪scraping_FanGraphs(url[0], None, url[1])])
player_plate_discipline
```

[12]:

```
      # Season            Name O-Swing% Z-Swing%  Swing% O-Contact%  \
0     1   2010    Scot Shields      0.0     0.0%    0.0%        0.0
1     2   2010  Brian Fuentes      0.0      0.0     0.0        0.0
2     3   2010 Fernando Rodney      0.0      0.0     0.0        0.0
3     4   2010       Dan Haren      0.0      0.0     0.0        0.0
4     5   2010   Ervin Santana      0.0      0.0     0.0        0.0
..  …    …             …      …      …      …      …
19  570   2012    Clay Hensley   100.0%    50.0%   66.7%     100.0%
20  571   2013      Jean Machi   100.0%   100.0%  100.0%       0.0%
21  572   2014      Jean Machi   100.0%   100.0%  100.0%     100.0%
22  573   2018   Roberto Gomez   100.0%    42.9%   50.0%     100.0%
23  574   2021     Jay Jackson   100.0%     0.0%   50.0%     100.0%

   Z-Contact% Contact%  Zone% F-Strike% SwStr%   CStr%    CSW%  Year Team
0         0.0      0.0 100.0%    100.0%   0.0%  100.0%  100.0%  None  LAA
1         0.0      0.0    0.0       0.0    0.0     0.0     0.0  None  LAA
2         0.0      0.0    0.0       0.0    0.0     0.0     0.0  None  LAA
3         0.0      0.0    0.0       0.0    0.0     0.0     0.0  None  LAA
4         0.0      0.0    0.0       0.0    0.0     0.0     0.0  None  LAA
..          …        …      …       …      …       …      …     …    …
19     100.0%   100.0%  66.7%    100.0%   0.0%   33.3%   33.3%  None  SFG
20      50.0%    33.3%  66.7%    100.0%  66.7%    0.0%   66.7%  None  SFG
21     100.0%   100.0%  50.0%    100.0%   0.0%    0.0%    0.0%  None  SFG
22      33.3%    50.0%  87.5%    100.0%  25.0%   50.0%   75.0%  None  SFG
23        0.0   100.0%  50.0%    100.0%   0.0%   50.0%   50.0%  None  SFG

[18060 rows x 16 columns]
```

Merging Dataframes

[115]:

```
# Removing Unnecessary Columns
player_standard = player_standard.drop(columns=['#','Year'], errors='ignore')
player_advanced = player_advanced.drop(columns=['#','Year'], errors='ignore')
player_batted = player_batted.drop(columns=['#','Year'], errors='ignore')
player_statcast = player_statcast.drop(columns=['#','Year'], errors='ignore')
player_plate_discipline = player_plate_discipline.drop(columns=['#','Year'],␣
↪errors='ignore')

player_table = pd.merge(player_standard, player_advanced, on=['Season', 'Name',␣
↪'PA', 'AVG', 'Team'])
player_table = pd.merge(player_table, player_batted, on=['Season', 'Name',␣
↪'Team', 'BABIP'])
```

```
player_table = pd.merge(player_table, player_statcast, on=['Season', 'Name', 'Team', 'AVG', 'PA', 'SLG', 'wOBA'])
player_table = pd.merge(player_table, player_plate_discipline, on=['Season', 'Name', 'Team'])

player_table = player_table.rename(columns={'Season':'Year'})

player_table = player_table.drop_duplicates()

# Removing the % in the values so that they can be used as numbers
player_table['BB%'] = player_table['BB%'].replace({'\%':''}, regex = True)
player_table['K%'] = player_table['K%'].replace({'\%':''}, regex = True)
player_table['LD%'] = player_table['BB%'].replace({'\%':''}, regex = True)
player_table['GB%'] = player_table['GB%'].replace({'\%':''}, regex = True)
player_table['FB%'] = player_table['FB%'].replace({'\%':''}, regex = True)
player_table['HR/FB'] = player_table['HR/FB'].replace({'\%':''}, regex = True)
player_table['Pull%'] = player_table['Pull%'].replace({'\%':''}, regex = True)
player_table['Cent%'] = player_table['Cent%'].replace({'\%':''}, regex = True)
player_table['Oppo%'] = player_table['Oppo%'].replace({'\%':''}, regex = True)
player_table['Soft%'] = player_table['Soft%'].replace({'\%':''}, regex = True)
player_table['Med%'] = player_table['Med%'].replace({'\%':''}, regex = True)
player_table['Hard%'] = player_table['Hard%'].replace({'\%':''}, regex = True)
player_table['Barrel%'] = player_table['Barrel%'].replace({'\%':''}, regex = True)
player_table['HardHit%'] = player_table['HardHit%'].replace({'\%':''}, regex = True)
player_table['O-Swing%'] = player_table['O-Swing%'].replace({'\%':''}, regex = True)
player_table['Z-Swing%'] = player_table['Z-Swing%'].replace({'\%':''}, regex = True)
player_table['Swing%'] = player_table['Swing%'].replace({'\%':''}, regex = True)
player_table['O-Contact%'] = player_table['O-Contact%'].replace({'\%':''}, regex = True)
player_table['Z-Contact%'] = player_table['Z-Contact%'].replace({'\%':''}, regex = True)
player_table['Contact%'] = player_table['Contact%'].replace({'\%':''}, regex = True)
player_table['Zone%'] = player_table['Zone%'].replace({'\%':''}, regex = True)
player_table['F-Strike%'] = player_table['F-Strike%'].replace({'\%':''}, regex = True)
player_table['SwStr%'] = player_table['SwStr%'].replace({'\%':''}, regex = True)
player_table['CStr%'] = player_table['CStr%'].replace({'\%':''}, regex = True)
player_table['CSW%'] = player_table['CSW%'].replace({'\%':''}, regex = True)

player_table = player_table.apply(pd.to_numeric, errors='ignore')
```

```python
player_table['EV'] = player_table['EV'].replace(0.0, np.NaN)
player_table['LA'] = player_table['LA'].replace(0.0, np.NaN)
player_table['Barrel%'] = player_table['Barrel%'].replace(0.0, np.NaN)

player_table = player_table.sort_values(by='Year')
player_table
```

[115]:

```
       Year             Name    G   AB   PA   H  1B  2B  3B  HR   R  RBI  BB  \
1230   2010    Clay Buchholz   28    1    1   1   1   0   0   0   0    0   0
19521  2010      Brandon Moss  17   26   27   4   3   1   0   0   2    2   1
4721   2010       Ben Revere   13   28   30   5   5   0   0   0   1    2   2
19528  2010  Jason Jaramillo   33   87   97  13  10   2   0   1   2    6   8
9464   2010        Mike McCoy  46   82   90  16  12   4   0   0   9    3   8
…      …                …    …    …    …  ..  ..  ..  ..  ..  ..    …  ..
3369   2021      Nomar Mazara  50  165  181  35  25   5   2   3  12   19  15
9424   2021     Cavan Biggio   79  250  294  56  38  10   1   7  27   27  37
3371   2021   Dustin Garneau   20   62   68  13   2   5   0   6   9   11   3
3355   2021      Willi Castro 125  413  450  91  61  15   6   9  56   38  23
25432  2021      Camilo Doval  29    0    0   0   0   0   0   0   0    0   0

       IBB   SO  HBP  SF  SH  GDP  SB  CS    AVG Team   BB%   K%  BB/K  \
1230     0    0    0   0   0    0   0   0  1.000  BOS   0.0   0.0  0.00
19521    0    6    0   0   0    1   0   0  0.154  PIT   3.7  22.2  0.17
4721     0    5    0   0   0    1   0   1  0.179  MIN   6.7  16.7  0.40
19528    1   14    1   1   0    7   0   0  0.149  PIT   8.2  14.4  0.57
9464     0   20    0   0   0    0   5   1  0.195  TOR   8.9  22.2  0.40
…      …    …    …  ..  ..    …  ..  ..    …    …     …     …
3369     0   45    0   1   0    4   0   0  0.212  DET   8.3  24.9  0.33
9424     2   78    1   4   1    4   3   1  0.224  TOR  12.6  26.5  0.47
3371     0   18    1   2   0    2   0   0  0.210  DET   4.4  26.5  0.17
3355     1  109    8   3   3    5   9   4  0.220  DET   5.1  24.2  0.21
25432    0    0    0   0   0    0   0   0  0.000  SFG   0.0   0.0  0.00

         OBP    SLG    OPS    ISO  Spd  BABIP   UBR  wGDP   wSB  wRC   wRAA  \
1230   1.000  1.000  2.000  0.000  0.1  1.000   0.0   0.0   0.0    1    0.5
19521  0.185  0.192  0.377  0.038  2.0  0.200  -0.4   0.0   0.0    0   -3.2
4721   0.233  0.179  0.412  0.000  1.6  0.217   0.0  -0.3  -0.4    0   -3.0
19528  0.227  0.207  0.434  0.057  0.1  0.164   0.4  -1.2  -0.1    2   -9.4
9464   0.267  0.244  0.511  0.049  5.5  0.258   0.6   0.3   0.5    4   -6.0
…        …      …      …      …    …      …     …     …     …
3369   0.276  0.321  0.597  0.109  2.7  0.271  -0.1   0.3  -0.1   14   -7.6
9424   0.322  0.356  0.678  0.132  3.7  0.290  -1.5   0.3  -0.1   32   -4.0
3371   0.250  0.581  0.831  0.371  1.4  0.175  -0.1  -0.4   0.0    9    1.2
3355   0.273  0.351  0.624  0.131  6.9  0.275   1.6   0.6  -0.2   38  -16.3
25432  0.000  0.000  0.000  0.000  0.1  0.000   0.0   0.0   0.0    0    0.0

        wOBA   wRC+  GB/FB   LD%    GB%    FB%  IFFB%  HR/FB  IFH   IFH%  BUH  \
```

1230	0.895	483.0	1.00	0.0	100.0	0.0	0.0%	0.0	0	0.0%	0
19521	0.172	0.0	1.83	3.7	55.0	30.0	0.0%	0.0	2	18.2%	0
4721	0.196	12.0	3.75	6.7	68.2	18.2	0.0%	0.0	1	6.7%	0
19528	0.200	18.0	1.32	8.2	50.0	37.8	17.9%	3.6	1	2.7%	0
9464	0.238	40.0	0.93	8.9	42.4	45.8	11.1%	0.0	2	8.0%	0
…	…	…	…	…	…	…	…	…	…	…	…
3369	0.264	64.0	1.49	8.3	47.9	32.2	15.4%	7.7	3	5.2%	0
9424	0.298	84.0	0.94	12.6	37.7	40.0	5.7%	10.0	1	1.5%	1
3371	0.335	113.0	0.73	4.4	34.8	47.8	22.7%	27.3	0	0.0%	0
3355	0.271	69.0	1.41	5.1	46.4	32.9	15.0%	9.0	7	5.0%	1
25432	0.000	0.0	0.00	0.0	0.0	0.0	0.0%	0.0	0	0.0%	0

	BUH%	Pull%	Cent%	Oppo%	Soft%	Med%	Hard%	Events	EV	maxEV \
1230	0.0%	100.0	0.0	0.0	0.0	100.0	0.0	0	NaN	0.0
19521	0.0%	35.0	50.0	15.0	15.0	45.0	40.0	0	NaN	0.0
4721	0.0%	39.1	34.8	26.1	26.1	47.8	26.1	0	NaN	0.0
19528	0.0%	33.8	39.2	27.0	24.3	56.8	18.9	0	NaN	0.0
9464	0.0%	33.9	41.9	24.2	19.4	54.8	25.8	0	NaN	0.0
…	…	…	…	…	…	…	…	…	…	…
3369	0.0%	33.9	37.2	28.9	14.0	56.2	29.8	121	90.4	111.5
9424	50.0%	35.0	35.6	29.4	10.7	58.8	30.5	178	88.9	109.6
3371	0.0%	56.5	26.1	17.4	26.1	41.3	32.6	46	86.4	106.4
3355	16.7%	32.3	32.3	35.5	22.3	55.8	21.9	310	85.6	115.4
25432	0.0%	0.0	0.0	0.0	0.0	0.0	0.0	0	NaN	0.0

	LA	Barrels	Barrel%	HardHit	HardHit%	xBA	xSLG	xwOBA	O-Swing% \
1230	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	0.0
19521	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	33.3
4721	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	27.0
19528	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	27.4
9464	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	19.5
…	…	…	…	…	…	…	…	…	…
3369	11.1	9.0	7.4	48.0	39.7	0.0	0.0	0.0	33.6
9424	15.4	10.0	5.6	56.0	31.5	0.0	0.0	0.0	22.2
3371	19.2	5.0	10.9	19.0	41.3	0.0	0.0	0.0	31.6
3355	11.4	16.0	5.2	91.0	29.4	0.0	0.0	0.0	42.4
25432	NaN	0.0	NaN	0.0	0.0	0.0	0.0	0.0	0.0

	Z-Swing%	Swing%	O-Contact%	Z-Contact%	Contact%	Zone%	F-Strike% \
1230	100.0	100.0	0.0	100.0	100.0	100.0	100.0
19521	72.3	52.0	70.6	91.2	84.3	48.0	51.9
4721	59.5	40.0	76.5	88.0	83.3	40.0	46.7
19528	61.4	43.3	78.8	92.2	87.7	46.6	68.0
9464	54.5	37.2	59.4	84.6	78.0	50.5	55.6
…	…	…	…	…	…	…	…
3369	65.4	46.6	62.7	86.1	76.2	41.1	62.4
9424	64.9	41.3	50.7	87.5	76.6	44.8	57.8

```
3371         74.1     48.6          65.5          77.9        73.0    40.0         52.9
3355         76.4     56.3          56.1          86.2        72.8    40.7         67.6
25432         0.0      0.0           0.0           0.0         0.0     0.0          0.0

       SwStr%  CStr%  CSW%
1230      0.0    0.0   0.0
19521     8.2   12.2  20.4
4721      6.7   21.9  28.6
19528     5.2   19.9  25.1
9464      8.3   21.2  29.5
…         …      …     …
3369     11.1   16.2  27.3
9424      9.7   20.2  29.9
3371     13.1   14.8  27.9
3355     15.3   12.3  27.6
25432     0.0    0.0   0.0

[17611 rows x 77 columns]
```

Removing Suffix From Player Names In order to match the names in the Lahman dataset, which we will take advantage of later, we will remove the suffix from player names. For example, as you will see below, Cedric Mullins is recorded as Cedric Mullins II on Fangraphs, but he is recorded as Cedric Mullins in the Lahman dataset.

[116]:
```
player_table[player_table.Name == 'Cedric Mullins II']
```

[116]:
```
     Year               Name    G   AB   PA    H   1B  2B  3B  HR   R  RBI  \
784  2018  Cedric Mullins II   45  170  191   40   27   9   0   4  23   11
910  2019  Cedric Mullins II   22   64   74    6    4   0   2   0   7    4
700  2020  Cedric Mullins II   48  140  153   38   28   4   3   3  16   12
658  2021  Cedric Mullins II  159  602  675  175  103  37   5  30  91   59

     BB  IBB   SO  HBP  SF  SH  GDP  SB  CS    AVG Team  BB%    K%  BB/K  \
784  17    0   37    2   0   2    1   2   3  0.235  BAL  8.9  19.4  0.46
910   4    0   14    3   1   2    2   1   0  0.094  BAL  5.4  18.9  0.29
700   8    0   37    1   0   4    0   7   2  0.271  BAL  5.2  24.2  0.22
658  59    3  125    8   4   1    2  30   8  0.291  BAL  8.7  18.5  0.47

       OBP    SLG    OPS    ISO  Spd  BABIP  UBR  wGDP  wSB  wRC  wRAA   wOBA  \
784  0.312  0.359  0.671  0.124  2.9  0.279  0.8   0.2 -0.9   20  -2.7  0.298
910  0.181  0.156  0.337  0.063  7.9  0.118  0.5  -0.2  0.2   -1 -10.3  0.159
700  0.315  0.407  0.723  0.136  7.2  0.350  1.8   0.6  0.4   18  -0.9  0.313
658  0.360  0.518  0.878  0.228  6.1  0.322  0.4   2.3  2.1  114  32.0  0.372

      wRC+  GB/FB  LD%   GB%   FB%  IFFB%  HR/FB  IFH   IFH%  BUH   BUH%  \
784   86.0   1.37  8.9  50.8  37.1  10.9%    8.7    9  14.3%    4  36.4%
910  -12.0   1.35  5.4  52.9  39.2  25.0%    0.0    1   3.7%    0   0.0%
```

```
700   95.0   1.25  5.2  43.5  34.8  21.9%    9.4    3   7.5%     9  60.0%
658  136.0   0.95  8.7  39.0  41.1  12.4%   15.5   17   9.2%     5  50.0%

     Pull%  Cent%  Oppo%  Soft%  Med%  Hard%  Events    EV  maxEV    LA  \
784   42.2   33.3   24.4   19.3  54.1   26.7     135  89.3  108.0  10.1
910   43.4   37.7   18.9   34.0  49.1   17.0      53  84.2  110.3  14.9
700   43.0   28.0   29.0   15.9  62.6   21.5     107  88.6  110.2  15.6
658   43.6   32.4   24.1   14.9  51.9   33.2     483  89.4  109.7  14.8

     Barrels  Barrel%  HardHit  HardHit%  xBA  xSLG  xwOBA  O-Swing%  \
784      4.0      3.0     38.0      28.1  0.0   0.0    0.0      22.2
910      1.0      1.9      9.0      17.0  0.0   0.0    0.0      33.9
700      3.0      2.8     34.0      31.8  0.0   0.0    0.0      33.0
658     39.0      8.1    189.0      39.1  0.0   0.0    0.0      30.0

     Z-Swing%  Swing%  O-Contact%  Z-Contact%  Contact%  Zone%  F-Strike%  \
784      64.4    40.6        66.7        90.3      83.1   43.8       58.1
910      62.0    45.4        64.4        85.3      76.1   41.0       66.2
700      68.1    48.0        70.2        85.1      79.2   42.7       59.5
658      64.5    45.1        71.7        87.8      81.7   43.6       59.1

     SwStr%  CStr%  CSW%
784     6.9   19.1  26.0
910    10.8   16.9  27.8
700    10.0   14.6  24.6
658     8.2   17.3  25.5
```

[117]:
```python
names = player_table.Name.str.split(' ', expand=True)[[0, 1]]
names.columns = ['First', 'Last']
names = names.assign(Name = names.First.str.cat(names.Last,sep=' '))
names = names[['Name']]

player_table = player_table.assign(Name = names.Name.to_list())
player_table[player_table.Name == 'Cedric Mullins']
```

[117]:
```
     Year            Name    G   AB   PA    H   1B  2B  3B  HR   R  RBI  BB  \
784  2018  Cedric Mullins   45  170  191   40   27   9   0   4  23   11  17
910  2019  Cedric Mullins   22   64   74    6    4   0   2   0   7    4   4
700  2020  Cedric Mullins   48  140  153   38   28   4   3   3  16   12   8
658  2021  Cedric Mullins  159  602  675  175  103  37   5  30  91   59  59

     IBB   SO  HBP  SF  SH  GDP  SB  CS    AVG Team  BB%    K%  BB/K    OBP  \
784    0   37    2   0   2    1   2   3  0.235  BAL  8.9  19.4  0.46  0.312
910    0   14    3   1   2    2   1   0  0.094  BAL  5.4  18.9  0.29  0.181
700    0   37    1   0   4    0   7   2  0.271  BAL  5.2  24.2  0.22  0.315
658    3  125    8   4   1    2  30   8  0.291  BAL  8.7  18.5  0.47  0.360
```

```
       SLG    OPS    ISO  Spd  BABIP  UBR  wGDP  wSB  wRC  wRAA   wOBA   wRC+  \
784  0.359  0.671  0.124  2.9  0.279  0.8   0.2 -0.9   20  -2.7  0.298   86.0
910  0.156  0.337  0.063  7.9  0.118  0.5  -0.2  0.2   -1 -10.3  0.159  -12.0
700  0.407  0.723  0.136  7.2  0.350  1.8   0.6  0.4   18  -0.9  0.313   95.0
658  0.518  0.878  0.228  6.1  0.322  0.4   2.3  2.1  114  32.0  0.372  136.0

     GB/FB  LD%   GB%   FB%  IFFB%  HR/FB  IFH   IFH%  BUH   BUH%  Pull%  \
784   1.37  8.9  50.8  37.1  10.9%    8.7    9  14.3%    4  36.4%   42.2
910   1.35  5.4  52.9  39.2  25.0%    0.0    1   3.7%    0   0.0%   43.4
700   1.25  5.2  43.5  34.8  21.9%    9.4    3   7.5%    9  60.0%   43.0
658   0.95  8.7  39.0  41.1  12.4%   15.5   17   9.2%    5  50.0%   43.6

     Cent%  Oppo%  Soft%  Med%  Hard%  Events    EV  maxEV    LA  Barrels  \
784   33.3   24.4   19.3  54.1   26.7     135  89.3  108.0  10.1      4.0
910   37.7   18.9   34.0  49.1   17.0      53  84.2  110.3  14.9      1.0
700   28.0   29.0   15.9  62.6   21.5     107  88.6  110.2  15.6      3.0
658   32.4   24.1   14.9  51.9   33.2     483  89.4  109.7  14.8     39.0

     Barrel%  HardHit  HardHit%  xBA  xSLG  xwOBA  O-Swing%  Z-Swing%  Swing%  \
784      3.0     38.0      28.1  0.0   0.0    0.0      22.2      64.4    40.6
910      1.9      9.0      17.0  0.0   0.0    0.0      33.9      62.0    45.4
700      2.8     34.0      31.8  0.0   0.0    0.0      33.0      68.1    48.0
658      8.1    189.0      39.1  0.0   0.0    0.0      30.0      64.5    45.1

     O-Contact%  Z-Contact%  Contact%  Zone%  F-Strike%  SwStr%  CStr%  CSW%
784        66.7        90.3      83.1   43.8       58.1     6.9   19.1  26.0
910        64.4        85.3      76.1   41.0       66.2    10.8   16.9  27.8
700        70.2        85.1      79.2   42.7       59.5    10.0   14.6  24.6
658        71.7        87.8      81.7   43.6       59.1     8.2   17.3  25.5
```

As you can see, the dataset I created from Fangraphs now has only first and last name in the Name column.

Add Player ID to Player Table from Lahman Dataset

[118]:
```python
player_info = pd.read_csv('tables/People.csv')
player_info = player_info[['playerID', 'nameFirst', 'nameLast']]
player_info = player_info.assign(Name = player_info.nameFirst.str.cat(player_info.nameLast,sep=' '))
player_info = player_info[['playerID', 'Name']]

player_table = pd.merge(player_table, player_info, on=['Name'])
```

Add Player's Position to Table

[119]:
```python
player_pos = pd.read_csv('tables/Appearances.csv')
player_pos = player_pos[['playerID', 'yearID', 'G_p', 'G_c', 'G_1b', 'G_2b', 'G_3b', 'G_ss', 'G_lf', 'G_cf', 'G_rf', 'G_dh']]
```

```
player_pos = player_pos[player_pos.yearID > 2010]
player_pos = player_pos.rename(columns = {'yearID':'Year', 'G_p':'P', 'G_c':
 ↪'C', 'G_1b':'1B', 'G_2b':'2B', 'G_3b':'3B', 'G_ss':'SS', 'G_lf':'LF', 'G_cf':
 ↪'CF', 'G_rf':'RF', 'G_dh':'DH'})
player_pos = player_pos.astype({'DH':'int32'})

positions = player_pos[['P', 'C', '1B', '2B', '3B', 'SS', 'LF', 'CF', 'RF',␣
 ↪'DH']]

positions = positions.assign(Pos = positions.idxmax(axis=1))

player_pos = pd.merge(player_pos, positions, on = ['P', 'C', '1B', '2B', '3B',␣
 ↪'SS', 'LF', 'CF', 'RF', 'DH'])
player_pos = player_pos[['playerID', 'Year', 'Pos']]
player_pos = player_pos.drop_duplicates()

player_table = pd.merge(player_table, player_pos, on=['playerID', 'Year'])
```

Reorder Columns of Dataframe

[120]:
```
player_table = player_table[['Year', 'Name', 'Team', 'Pos', 'G', 'AB', 'PA',␣
 ↪'HR', 'R', 'RBI', 'BB', 'SO', 'AVG', 'BB/K', 'OBP', 'SLG', 'OPS', 'ISO',␣
 ↪'wOBA', 'wRC+', 'EV', 'LA', 'Barrel%', 'O-Swing%', 'Z-Swing%', 'Swing%',␣
 ↪'SwStr%', 'Contact%']]

player_table
```

[120]:
```
       Year             Name Team Pos   G  AB  PA  HR  R  RBI  BB  SO    AVG  \
0      2011   Clay Buchholz  BOS   P  14   0   0   0  0    0   0   0  0.000
1      2012   Clay Buchholz  BOS   P  29   2   3   0  0    0   0   1  0.000
2      2013   Clay Buchholz  BOS   P  16   0   0   0  0    0   0   0  0.000
3      2014   Clay Buchholz  BOS   P  28   2   2   0  0    0   0   0  0.500
4      2015   Clay Buchholz  BOS   P  18   6   6   0  0    0   0   2  0.000
…      …                …   …  ..  ..  ..  ..  .. ..   …  ..  ..    …
16353  2021      Joe Barlow  TEX   P  31   0   0   0  0    0   0   0  0.000
16354  2021     Glenn Otto  TEX   P   6   0   0   0  0    0   0   0  0.000
16355  2021    Kevin Smith  TOR  3B  18  32  36   1  2    1   3  11  0.094
16356  2021  Josh Palacios  TOR  RF  13  35  42   0  7    4   3  11  0.200
16357  2021    Camilo Doval  SFG   P  29   0   0   0  0    0   0   0  0.000

       BB/K    OBP    SLG    OPS    ISO   wOBA   wRC+    EV    LA  Barrel%  \
0      0.00  0.000  0.000  0.000  0.000  0.000    0.0   NaN   NaN      NaN
1      0.00  0.000  0.000  0.000  0.000  0.000 -100.0   NaN   NaN      NaN
2      0.00  0.000  0.000  0.000  0.000  0.000    0.0   NaN   NaN      NaN
3      0.00  0.500  0.500  1.000  0.000  0.446  187.0   NaN   NaN      NaN
4      0.00  0.000  0.000  0.000  0.000  0.000 -100.0  89.4  14.7      NaN
…       …      …      …      …      …      …      …     …     …        …
```

```
16353  0.00  0.000  0.000  0.000  0.000  0.000   0.0   NaN   NaN      NaN
16354  0.00  0.000  0.000  0.000  0.000  0.000   0.0   NaN   NaN      NaN
16355  0.27  0.194  0.188  0.382  0.094  0.182   6.0  86.7  36.5     14.3
16356  0.27  0.293  0.200  0.493  0.000  0.236  42.0  93.5   8.4      3.8
16357  0.00  0.000  0.000  0.000  0.000  0.000   0.0   NaN   NaN      NaN

       O-Swing%  Z-Swing%  Swing%  SwStr%  Contact%
0           0.0       0.0     0.0     0.0       0.0
1          42.9      66.7    50.0    20.0      60.0
2           0.0       0.0     0.0     0.0       0.0
3          33.3      40.0    36.4     9.1      75.0
4          14.3      62.5    40.0    10.0      75.0
…           …       …     …      …       …
16353       0.0       0.0     0.0     0.0       0.0
16354       0.0       0.0     0.0     0.0       0.0
16355      35.3      85.5    57.7    16.3      71.8
16356      35.2      65.3    48.5    14.7      69.6
16357       0.0       0.0     0.0     0.0       0.0

[16358 rows x 28 columns]
```

Remove Pitchers from Dataset

[121]:
```python
player_table = player_table[player_table.Pos != 'P']
player_table
```

[121]:
```
       Year           Name Team Pos    G   AB   PA  HR   R  RBI  BB   SO  \
9      2011   Brandon Moss  PHI  RF    5    6    6   0   0    0   0    2
10     2012   Brandon Moss  OAK  1B   84  265  296  21  48   52  26   90
11     2013   Brandon Moss  OAK  1B  145  446  505  30  73   87  50  140
12     2014   Brandon Moss  OAK  1B  147  500  580  25  70   81  67  153
13     2015   Brandon Moss  STL  RF   51  132  151   4  11    8  17   42
…      …              …   …  ..   …   …   …   ..   ..  …   ..  …
16346  2021    Akil Baddoo  DET  CF  124  413  461  13  60   55  45  122
16348  2021     Zack Short  DET  SS   61  156  184   6  21   20  22   59
16351  2021     Ryan Dorow  TEX  3B    3    6    7   0   0    0   1    3
16355  2021    Kevin Smith  TOR  3B   18   32   36   1   2    1   3   11
16356  2021  Josh Palacios  TOR  RF   13   35   42   0   7    4   3   11

         AVG  BB/K    OBP    SLG    OPS    ISO   wOBA   wRC+    EV    LA  \
9      0.000  0.00  0.000  0.000  0.000  0.000  0.000 -100.0   NaN   NaN
10     0.291  0.29  0.358  0.596  0.954  0.306  0.402  160.0   NaN   NaN
11     0.256  0.36  0.337  0.522  0.859  0.267  0.369  137.0   NaN   NaN
12     0.234  0.44  0.334  0.438  0.772  0.204  0.339  122.0   NaN   NaN
13     0.250  0.40  0.344  0.409  0.753  0.159  0.328  109.0  88.2  17.9
…        …   …     …     …     …     …     …     …    …    …
16346  0.259  0.37  0.330  0.436  0.766  0.177  0.329  108.0  86.0  13.8
```

```
16348  0.141  0.37  0.239  0.282  0.521  0.141  0.230   41.0  87.5  24.3
16351  0.000  0.33  0.143  0.000  0.143  0.000  0.099  -46.0  88.5  29.6
16355  0.094  0.27  0.194  0.188  0.382  0.094  0.182    6.0  86.7  36.5
16356  0.200  0.27  0.293  0.200  0.493  0.000  0.236   42.0  93.5   8.4

       Barrel%  O-Swing%  Z-Swing%  Swing%  SwStr%  Contact%
9          NaN      54.5      85.7    66.7    27.8      58.3
10         NaN      35.0      72.0    50.5    16.5      67.0
11         NaN      35.6      71.7    49.8    14.6      70.5
12         NaN      32.8      67.6    47.0    12.4      73.3
13        10.0      31.6      69.4    46.5    13.0      71.6
…          …       …       …       …      …       …
16346      8.8      27.7      70.4    45.9    12.9      72.0
16348      4.9      22.8      66.0    41.7    10.5      74.9
16351      NaN      40.0      66.7    54.5    20.8      58.3
16355     14.3      35.3      85.5    57.7    16.3      71.8
16356      3.8      35.2      65.3    48.5    14.7      69.6

[7894 rows x 28 columns]
```

Add Player Salaries to Dataset

[122]:
```python
salaries = pd.read_csv('salaries/salaries.csv')

# Split the Name colum into first name and last name (originally stored as 'Last, First')
# and store it as a separate dataframe
names = salaries.Player.str.split(', ', expand=True)[[0, 1]]

# Create a new column called Name that has the format 'First Last'
names = names.assign(Name = names[1].str.cat(names[0],sep=' '))

# Remove all columns except for the new name column
names = names[['Name']]

# Add the years to the names dataframe
names = names.assign(Year = salaries.Year.to_list())

# Add the salaries to the names dataframe
names = names.assign(Salary = salaries.Salary.to_list())

names.Salary = names.Salary.str.replace(',', '')
names.Salary = names.Salary.replace({'\$':''}, regex = True)

# Assign names to the salaries variable
salaries = names
```

```python
player_table = pd.merge(player_table, salaries, on = ['Name', 'Year'])
player_table
```

[122]:

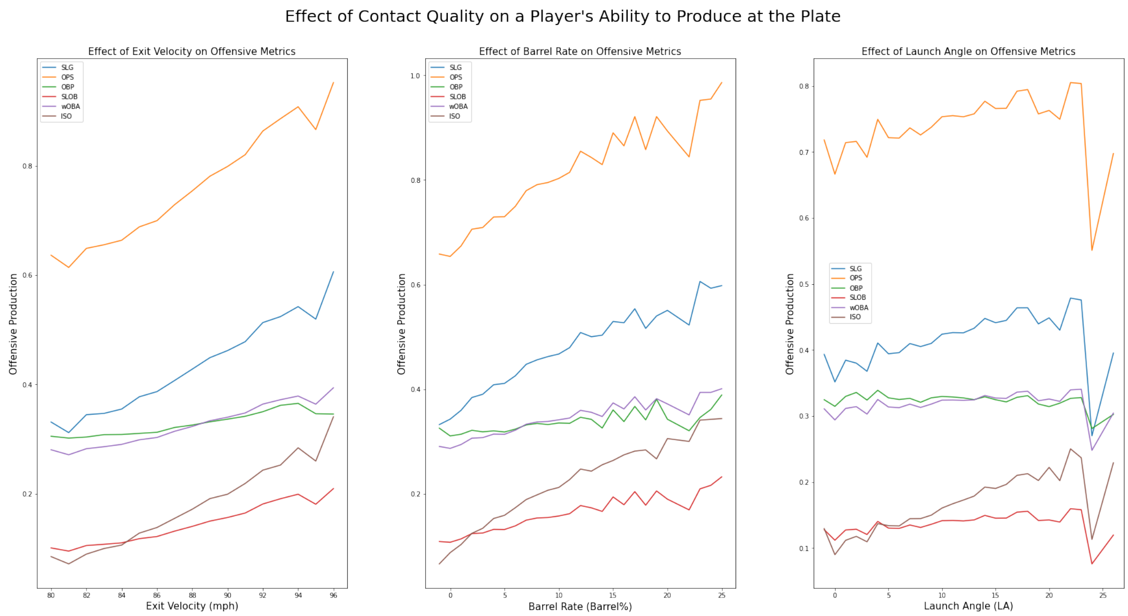
```
      Year              Name Team Pos    G   AB   PA  HR   R  RBI  BB   SO  \
0     2013      Brandon Moss  OAK  1B  145  446  505  30  73   87  50  140
1     2014      Brandon Moss  OAK  1B  147  500  580  25  70   81  67  153
2     2015      Brandon Moss  STL  RF   51  132  151   4  11    8  17   42
3     2015      Brandon Moss  STL  1B   51  132  151   4  11    8  17   42
4     2015      Brandon Moss  CLE  RF   94  337  375  15  36   50  32  106
…      …                 …    …  ..    …    …    …  ..  ..    …  ..    …
5648  2021   Geraldo Perdomo  ARI  SS   11   31   37   0   5    1   6    6
5649  2021  Stuart Fairchild  ARI  CF   12   15   17   0   3    2   1    3
5650  2021        Kyle Isbel  KCR  RF   28   76   83   1  16    7   7   23
5651  2021       Zack Short  DET  SS   61  156  184   6  21   20  22   59
5652  2021     Josh Palacios  TOR  RF   13   35   42   0   7    4   3   11

        AVG  BB/K    OBP    SLG    OPS    ISO   wOBA   wRC+    EV    LA  \
0     0.256  0.36  0.337  0.522  0.859  0.267  0.369  137.0   NaN   NaN
1     0.234  0.44  0.334  0.438  0.772  0.204  0.339  122.0   NaN   NaN
2     0.250  0.40  0.344  0.409  0.753  0.159  0.328  109.0  88.2  17.9
3     0.250  0.40  0.344  0.409  0.753  0.159  0.328  109.0  88.2  17.9
4     0.217  0.30  0.288  0.407  0.695  0.190  0.300   86.0  89.4  20.5
…         …     …      …      …      …      …      …      …     …     …
5648  0.258  1.00  0.378  0.419  0.798  0.161  0.331  104.0  86.5  18.7
5649  0.133  0.33  0.235  0.200  0.435  0.067  0.208   24.0  81.3  12.9
5650  0.276  0.30  0.337  0.434  0.772  0.158  0.333  109.0  87.3  19.0
5651  0.141  0.37  0.239  0.282  0.521  0.141  0.230   41.0  87.5  24.3
5652  0.200  0.27  0.293  0.200  0.493  0.000  0.236   42.0  93.5   8.4

      Barrel%  O-Swing%  Z-Swing%  Swing%  SwStr%  Contact%   Salary
0         NaN      35.6      71.7    49.8    14.6      70.5  1600000
1         NaN      32.8      67.6    47.0    12.4      73.3  4100000
2        10.0      31.6      69.4    46.5    13.0      71.6  6500000
3        10.0      31.6      69.4    46.5    13.0      71.6  6500000
4        12.0      33.0      74.6    50.8    14.1      72.0  6500000
…           …         …         …       …       …         …        …
5648      4.0      16.9      67.2    39.5    10.1      74.5   570500
5649      NaN      30.6      76.0    49.2    16.4      66.7   570500
5650      3.8      39.8      60.7    48.6    10.9      77.6   570500
5651      4.9      22.8      66.0    41.7    10.5      74.9   570500
5652      3.8      35.2      65.3    48.5    14.7      69.6   570500

[5653 rows x 29 columns]
```

0.0.4 Part III: Analyzing Offensive Metrics Using Player Data

Effect of Contact Quality on Production

[123]:
```
fig, ax = plt.subplots(1, 3)
fig.subplots_adjust(wspace=.25)
fig.set_figheight(15)
fig.set_figwidth(30)
fig.suptitle("Effect of Contact Quality on a Player's Ability to Produce at the␣
↪Plate", fontsize=25, y=.95)


players = player_table[player_table.PA > 200]
players = players[players.Year > 2014]
players = players.apply(pd.to_numeric, errors='ignore')

players = players.assign(SLOB = players.SLG * players.OBP)

######################################################################
#                       Plotting Exit Velocity                       #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['Velo'] = pd.cut(players.EV, velos, include_lowest=True, labels =␣
↪labels)

# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['Velo'] = players[['Velo']].apply(lambda col:pd.Categorical(col).codes)

players.groupby('Velo')[['SLG', 'OPS', 'OBP', 'SLOB', 'wOBA', 'ISO']].mean().
↪plot(legend=True, ax=ax[0])

# Label the title, x-axis, and y-axis
ax[0].set_title('Effect of Exit Velocity on Offensive Metrics', fontsize=15)
ax[0].set_xlabel("Exit Velocity (mph)", fontsize=15)
ax[0].set_ylabel("Offensive Production", fontsize=15)

plt.xlim([80,96])

######################################################################
#                       Plotting Barrel Rate                         #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['Barrel%'] = pd.cut(players['Barrel%'], velos, include_lowest=True,␣
↪labels = labels)
```

```
# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['Barrel%'] = players[['Barrel%']].apply(lambda col:pd.Categorical(col).
↪codes)

players.groupby('Barrel%')[['SLG', 'OPS', 'OBP', 'SLOB', 'wOBA', 'ISO']].mean().
↪plot(legend=True, ax=ax[1])

# Label the title, x-axis, and y-axis
ax[1].set_title('Effect of Barrel Rate on Offensive Metrics', fontsize=15)
ax[1].set_xlabel("Barrel Rate (Barrel%)", fontsize=15)
ax[1].set_ylabel("Offensive Production", fontsize=15)

######################################################################
#                       Plotting Launch Angle                        #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['LA'] = pd.cut(players['LA'], velos, include_lowest=True, labels =␣
↪labels)

# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['LA'] = players[['LA']].apply(lambda col:pd.Categorical(col).codes)

players.groupby('LA')[['SLG', 'OPS', 'OBP', 'SLOB', 'wOBA', 'ISO']].mean().
↪plot(ax=ax[2])

# Label the title, x-axis, and y-axis
ax[2].set_title('Effect of Launch Angle on Offensive Metrics', fontsize=15)
ax[2].set_xlabel("Launch Angle (LA)", fontsize=15)
ax[2].set_ylabel("Offensive Production", fontsize=15)
ax[2].legend(loc=(.05,.5))

plt.xlim([-2,27])

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

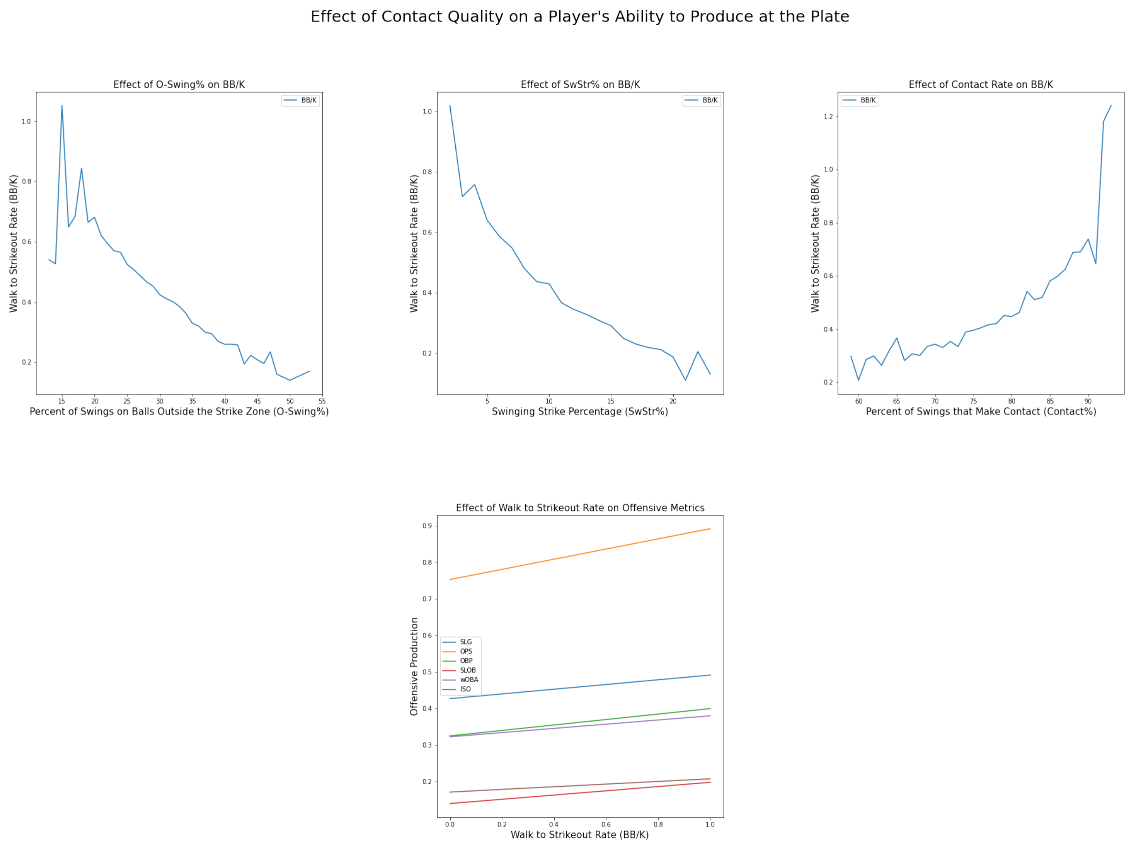


Effect of Plate Discipline on Production

[124]:
```
fig, ax = plt.subplots(2, 3)
fig.subplots_adjust(wspace=.25)
fig.set_figheight(20)
fig.set_figwidth(30)
fig.delaxes(ax[1,0])
fig.delaxes(ax[1,2])
fig.suptitle("Effect of Contact Quality on a Player's Ability to Produce at the Plate", fontsize=25, y=.99)


players = player_table[player_table.PA > 200]
players = players[players.Year > 2014]
players = players.apply(pd.to_numeric, errors='ignore')

players = players.assign(SLOB = players.SLG * players.OBP)

######################################################################
#                          Plotting O-Swing%                         #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['O-Swing%'] = pd.cut(players['O-Swing%'], velos, include_lowest=True, labels = labels)
```

```
# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['O-Swing%'] = players[['O-Swing%']].apply(lambda col:pd.
↪Categorical(col).codes)

players.groupby('O-Swing%')[['BB/K']].mean().plot(legend=True, ax=ax[0,0])

# Label the title, x-axis, and y-axis
ax[0,0].set_title('Effect of O-Swing% on BB/K', fontsize=15)
ax[0,0].set_xlabel("Percent of Swings on Balls Outside the Strike Zone␣
↪(O-Swing%)", fontsize=15)
ax[0,0].set_ylabel("Walk to Strikeout Rate (BB/K)", fontsize=15)

######################################################################
#                   Plotting Swinging Strike Percentage              #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['SwStr%'] = pd.cut(players['SwStr%'], velos, include_lowest=True,␣
↪labels = labels)

# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['SwStr%'] = players[['SwStr%']].apply(lambda col:pd.Categorical(col).
↪codes)

players.groupby('SwStr%')[['BB/K']].mean().plot(legend=True, ax=ax[0,1])

# Label the title, x-axis, and y-axis
ax[0,1].set_title('Effect of SwStr% on BB/K', fontsize=15)
ax[0,1].set_xlabel("Swinging Strike Percentage (SwStr%)", fontsize=15)
ax[0,1].set_ylabel("Walk to Strikeout Rate (BB/K)", fontsize=15)

######################################################################
#                          Plotting Contact%                         #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['Contact%'] = pd.cut(players['Contact%'], velos, include_lowest=True,␣
↪labels = labels)
```

```python
# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['Contact%'] = players[['Contact%']].apply(lambda col:pd.
↪Categorical(col).codes)

players.groupby('Contact%')[['BB/K']].mean().plot(legend=True, ax=ax[0,2])

# Label the title, x-axis, and y-axis
ax[0,2].set_title('Effect of Contact Rate on BB/K', fontsize=15)
ax[0,2].set_xlabel("Percent of Swings that Make Contact (Contact%)",␣
↪fontsize=15)
ax[0,2].set_ylabel("Walk to Strikeout Rate (BB/K)", fontsize=15)

######################################################################
#                         Plotting BB/K                              #
######################################################################
velos = [i for i in range(0, 100, 1)]
labels = [i for i in range(0, 99, 1)]

players['BB/K'] = pd.cut(players['BB/K'], velos, include_lowest=True, labels =␣
↪labels)

# Make categorical column (returned by pd.cut) into int -- https://
↪stackoverflow.com/questions/38088652/pandas-convert-categories-to-numbers/
↪61761109#61761109
players['BB/K'] = players[['BB/K']].apply(lambda col:pd.Categorical(col).codes)

players.groupby('BB/K')[['SLG', 'OPS', 'OBP', 'SLOB', 'wOBA', 'ISO']].mean().
↪plot(legend=True, ax=ax[1,1])

# Label the title, x-axis, and y-axis
ax[1,1].set_title('Effect of Walk to Strikeout Rate on Offensive Metrics',␣
↪fontsize=15)
ax[1,1].set_xlabel("Walk to Strikeout Rate (BB/K)", fontsize=15)
ax[1,1].set_ylabel("Offensive Production", fontsize=15)

plt.subplots_adjust(left=0.1,bottom=0.1, right=0.9, top=0.9, wspace=0.4,␣
↪hspace=0.4)

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



0.0.5 Part IV: Creating Player Projections for 2022

[125]:
```
def getPlayer(df, year, name):
    temp = df.drop(['Year'], axis=1)
    temp.insert(0,'Year', year)
    return temp[temp[name] == 1.0].head(1)
```

[168]:
```
players = player_table[player_table.Year != 2020]
players = players.assign(SLOB = players.SLG * players.OBP)
players = players[~players.isnull().any(axis=1)]
players = players[['Name', 'Year', 'G', 'AB', 'PA', 'HR', 'R', 'RBI', 'BB',␣
 ↪'SO', 'AVG', 'BB/K', 'OBP', 'SLG', 'OPS', 'ISO', 'wOBA', 'wRC+', 'EV', 'LA',␣
 ↪'Barrel%', 'O-Swing%', 'Z-Swing%', 'Swing%', 'SwStr%', 'Contact%']]

x = players[['Name', 'Year']]
y = players.drop(['Name', 'Year'], axis=1)

# One-Hot Encode the names of players
ohe = OneHotEncoder()
data = ohe.fit_transform(x[['Name']])
```

```
x[ohe.categories_[0]] = data.toarray()
x = x.drop(['Name'], axis=1)
```

[186]:

```
model = LinearRegression()
model.fit(x, y)

count = 0
for name in x:
    test = player_table[player_table.Name == name]
    if 2022 - test.Year.max() < 3:
        test = test[test.Year == test.Year.max()]
        pos = test.Pos.values[0]
        if count == 0:
            count = 1
        elif count == 1:
            predictions = pd.DataFrame(data=model.predict(getPlayer(x, 2022,␣
↪name)), columns=players.drop(['Name', 'Year', 'Pos'], axis=1).columns)
            predictions.insert(0, 'Name', name)
            predictions.insert(1, 'Year', 2022)
            predictions.insert(2, 'Pos', pos)
            count = 2
        else:
            temp = pd.DataFrame(data=model.predict(getPlayer(x, 2022, name)),␣
↪columns=players.drop(['Name', 'Year', 'Pos'], axis=1).columns)
            temp.insert(0, 'Name', name)
            temp.insert(1, 'Year', 2022)
            temp.insert(2, 'Pos', pos)
            predictions = pd.concat([predictions, temp])

predictions = predictions.round(decimals = 3)
predictions['G'] = predictions['G'].astype(int)
predictions['AB'] = predictions['AB'].astype(int)
predictions['PA'] = predictions['PA'].astype(int)
predictions['HR'] = predictions['HR'].astype(int)
predictions['R'] = predictions['R'].astype(int)
predictions['RBI'] = predictions['RBI'].astype(int)
predictions['BB'] = predictions['BB'].astype(int)
predictions['SO'] = predictions['SO'].astype(int)
predictions['wRC+'] = predictions['wRC+'].astype(int)

predictions
```

[186]:

```
                 Name  Year Pos    G   AB   PA  HR   R  RBI  BB   SO    AVG  \
0         Aaron Judge  2022  RF  111  394  481  34  82   74  80  151  0.265
0        Abraham Toro  2022  3B   41  144  162   4  19   20  13   23  0.227
0  Adalberto Mondesi  2022  3B   55  209  222   8  33   31   7   73  0.244
0         Adam Duvall  2022  LF   77  262  288  19  35   56  19   91  0.221
```

```
0          Adam Eaton  2022  RF   73  261  304   5  46   24  31   56  0.246
..                ...   ...  ..  ...  ...  ...  ..  ..  ...  ..  ...  ...
0      Yonathan Daza  2022  CF  101  277  305   1  22   27  19   56  0.277
0     Yordan Alvarez  2022  DH  138  513  572  32  88  101  48  141  0.272
0     Zach McKinstry  2022  RF   54  134  146   6  15   26   8   46  0.210
0       Zack Collins  2022   C   72  171  205   3  21   23  32   65  0.205
0         Zack Short  2022  SS   55  132  158   5  17   17  20   55  0.136

    BB/K    OBP    SLG    OPS    ISO   wOBA  wRC+     EV      LA  Barrel%  \
0  0.503  0.383  0.545  0.927  0.279  0.387   146  95.414  13.869   20.836
0  0.549  0.305  0.371  0.676  0.145  0.297    90  86.570  13.994    7.342
0  0.137  0.282  0.446  0.727  0.202  0.306    89  90.420  14.151   11.478
0  0.222  0.283  0.478  0.761  0.256  0.319    96  89.292  23.596   15.340
0  0.490  0.333  0.379  0.712  0.133  0.312    93  87.691  10.313    5.062
..   ...    ...    ...    ...    ...    ...   ...     ...     ...      ...
0  0.349  0.329  0.350  0.681  0.073  0.301    72  85.220   6.944    2.442
0  0.339  0.343  0.526  0.870  0.253  0.366   135  93.220  14.444   16.342
0  0.199  0.260  0.400  0.661  0.190  0.279    74  88.420  14.044    8.542
0  0.489  0.327  0.333  0.662  0.128  0.297    87  91.120  21.244   10.642
0  0.369  0.236  0.277  0.514  0.141  0.227    38  87.520  24.644    5.342

   O-Swing%  Z-Swing%  Swing%  SwStr%  Contact%
0    25.761    67.479  41.549  13.986    66.450
0    34.826    66.282  48.180   7.984    83.423
0    39.704    82.185  57.048  20.357    64.062
0    36.098    69.513  49.621  13.581    72.501
0    30.616    64.766  44.789   9.074    79.918
..      ...       ...     ...     ...       ...
0    36.526    69.932  50.030   9.734    80.623
0    30.426    62.432  43.030   9.234    78.623
0    33.826    61.532  46.030  10.934    76.223
0    23.226    70.032  43.630  11.634    73.423
0    22.926    66.332  41.730  10.734    74.423

[508 rows x 27 columns]
```

0.0.6 Part V: Making a Lineup Out of the Top Projected Performers

[189]:
```python
players[players.Name == 'Aaron Judge']
```

[189]:
```
             Name  Year Pos    G   AB   PA  HR    R  RBI   BB   SO    AVG  \
4658  Aaron Judge  2017  RF  155  542  678  52  128  114  127  208  0.284
4659  Aaron Judge  2018  RF  112  413  498  27   77   67   76  152  0.278
4660  Aaron Judge  2019  RF  102  378  447  27   75   55   64  141  0.272
4662  Aaron Judge  2021  RF  148  550  633  39   89   98   75  158  0.287

       BB/K    OBP    SLG    OPS    ISO   wOBA  wRC+     EV     LA  Barrel%  \
```

4658	0.61	0.422	0.627	1.049	0.343	0.430	174.0	94.9	15.8	24.9
4659	0.50	0.392	0.528	0.919	0.249	0.391	150.0	94.7	12.4	15.4
4660	0.45	0.381	0.540	0.921	0.267	0.382	141.0	96.0	11.2	19.7
4662	0.47	0.373	0.544	0.916	0.256	0.387	148.0	95.8	11.6	17.6

	O-Swing%	Z-Swing%	Swing%	SwStr%	Contact%
4658	24.7	66.2	41.1	13.3	67.6
4659	25.1	63.8	40.3	13.7	65.9
4660	24.6	68.1	41.9	14.6	65.1
4662	27.0	67.5	42.5	11.3	73.4

[ ]: