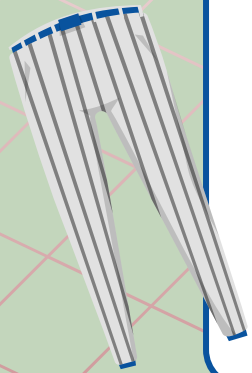
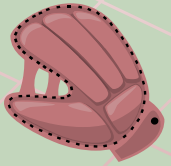
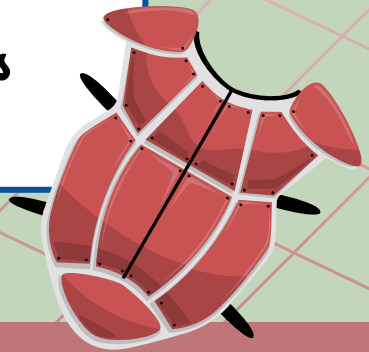
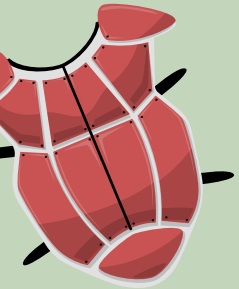




MLB Statistics

By: Zach Hracho, Minato Myers, and Eric Powers



Main Websites

STATCAST
POWERED BY Google Cloud

kaggle

```
#The baseballsavant website would not allow for multiple seasons to be downloaded at one with a max of two
#All of this data is pre-filtered into home run events in order to make the dataset managable and not too large
df = pd.read_csv('Datasets/Home_Run_Data.csv')
mergeDF = pd.read_csv('Datasets/MergedHR.csv')

#Built-in function which allows for the data in each csv to be appended onto one another, increasing the amount of data
ans = pd.concat([df,mergeDF], ignore_index = True)
ans.dropna(axis = 1, how = 'all', inplace = True)
ans.head(10)
```

What is Pandas?

	pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_name	batter	pitcher	events	description	...	fld_score
0	FF	2022-10-05	89.2	2.43	5.40	Vázquez, Christian	543877	674681	home_run	hit_into_play	...	0
1	SL	2022-10-05	91.1	-1.81	6.24	Moreno, Gabriel	672515	657508	home_run	hit_into_play	...	0
2	SI	2022-10-05	90.7	2.03	5.69	Torrens, Luis	620443	641302	home_run	hit_into_play	...	3
3	FF	2022-10-05	93.6	-1.58	6.33	Choi, Ji-Man	596847	601713	home_run	hit_into_play	...	3
4	CU	2022-10-05	78.5	-1.77	5.39	McCann, James	543510	607200	home_run	hit_into_play	...	0
5	FF	2022-10-05	91.2	-1.28	6.43	Vavra, Terrin	679631	669952	home_run	hit_into_play	...	4
6	CH	2022-10-05	81.1	2.99	5.26	Martinez, J.D.	502110	676596	home_run	hit_into_play	...	2
7	FC	2022-10-05	86.8	-1.94	5.73	Villar, David	681584	489334	home_run	hit_into_play	...	1
8	FC	2022-10-05	85.3	-0.78	6.43	Isbel, Kyle	664728	650644	home_run	hit_into_play	...	6

Matplotlib/ Seaborn

01

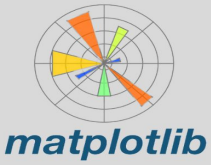
Seaborn is a data visualization library that works hand-in-hand with matplotlib

02

Similar to what was discussed in class, but allows for more flexible plotting/ graphical capabilities

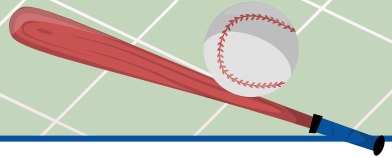
03

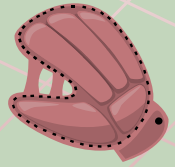
For the purpose of this project, matplotlib will be used to define the plot, margins, and axes while seaborn is used to plot the functions and data

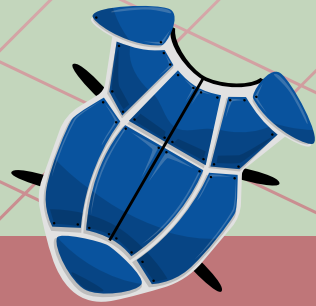


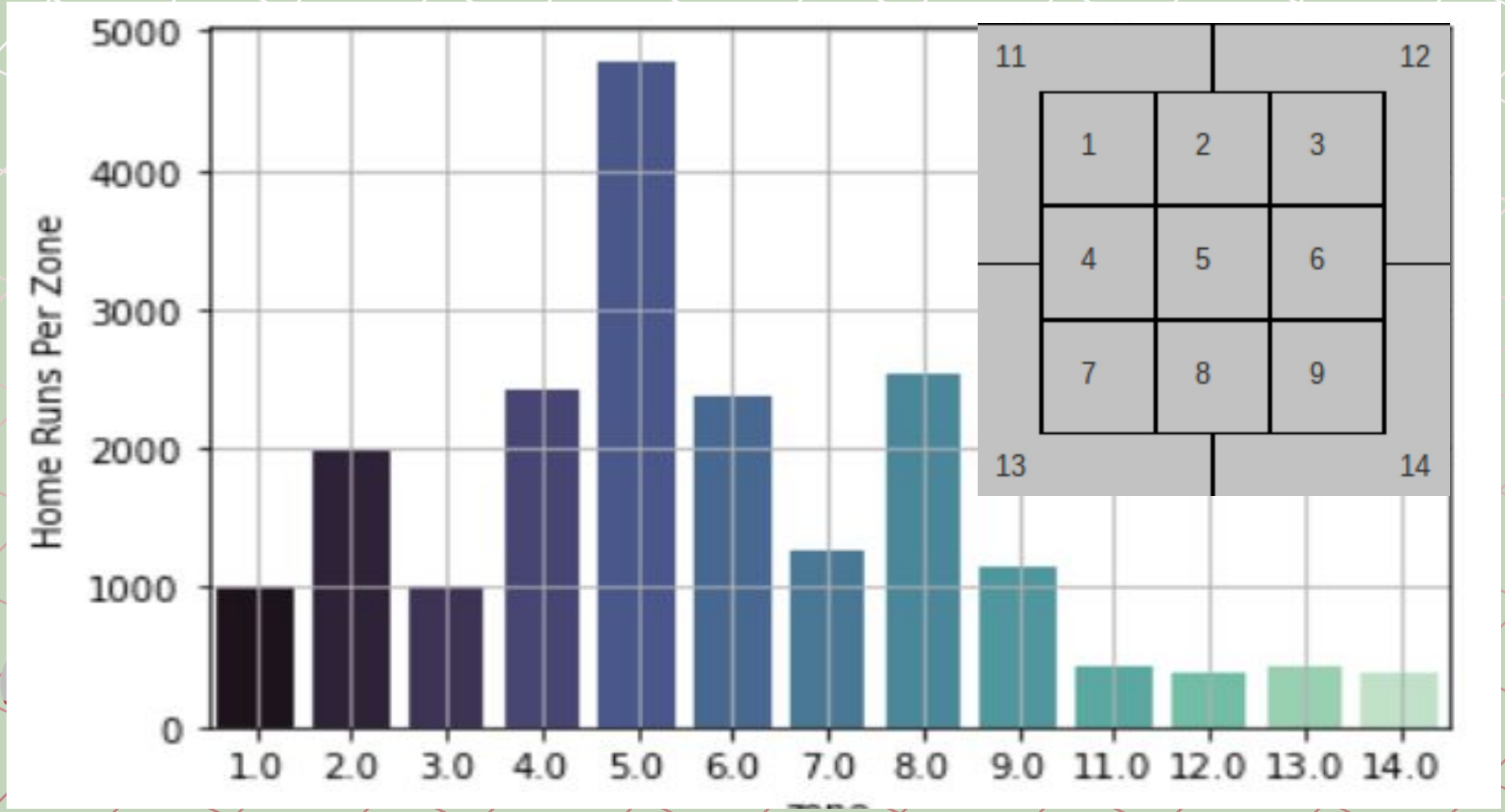
```
#Before we start, we want to see if there is a correlation between any variables
#Initially, there were too many variables to make the corr matrix readable, so we have to minimize
corrMatrix = round(ans.corr(), 2)
corrMatrix = corrMatrix.loc['release_speed':'plate_x']
corrMatrix = corrMatrix[corrMatrix.columns[0:8]]

#This correlation matrix will show the dependency of variables within different categories
#The close the value to one, the more related the variables are to one another
sns.heatmap(corrMatrix, annot = True, vmin = -1, vmax = 1, cmap ='Blues')
plt.figure(figsize = (25,10))
plt.show()
```

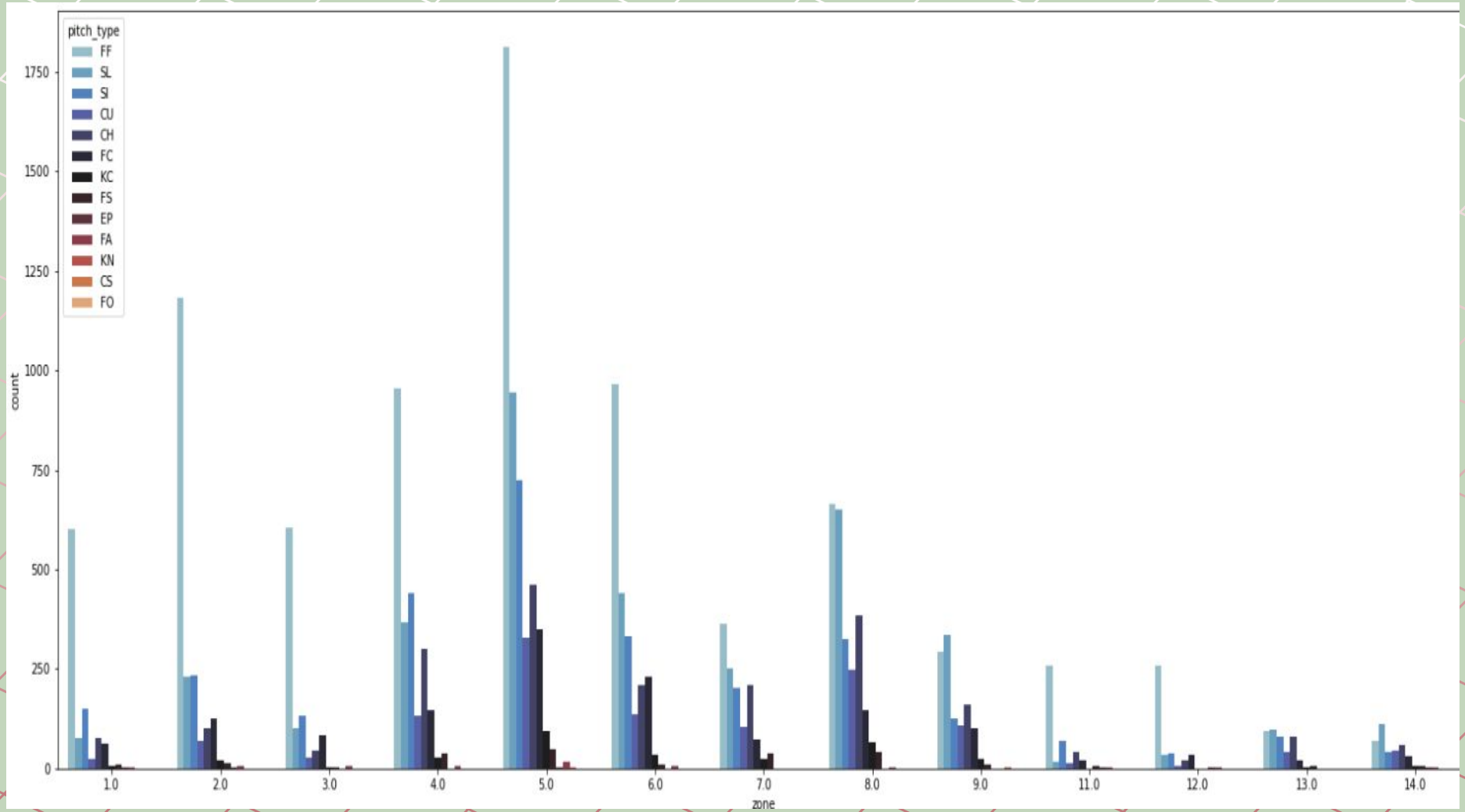
Graphs

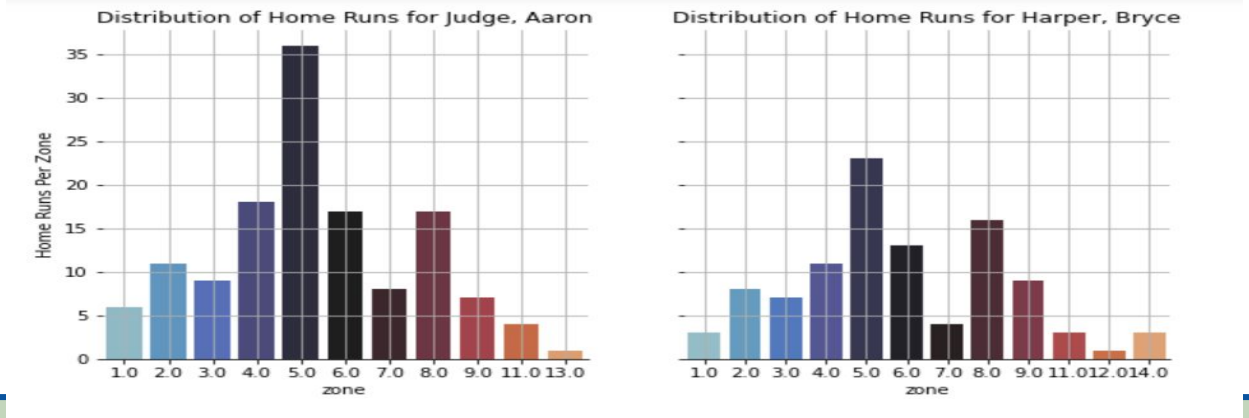




Home Runs Per Zone
5000
4000
3000
2000
1000
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
11.0
12.0
13.0
14.0
11
12
1
2
3
4
5
6
7
8
9
13
14


pitch_type
FF
SL
SI
CU
CH
FC
KC
FS
EP
FA
KN
CS
FO
count
1750
1500
1250
1000
750
500
250
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
11.0
12.0
13.0
14.0
zone

Comparing Two Players

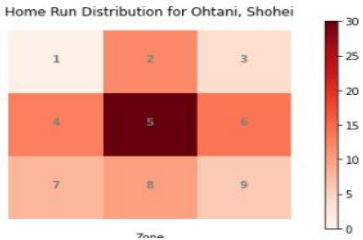
- We will now transition to Jupyter Notebook for an interactive demonstration of the graphical functions implemented in this project
  - The function comparePlayers takes three inputs: the pandas dataframe, and the two player names that we want to compare for home runs



- Wanted to ns for a specific pl
  - Requ
- Only analy ke zone)

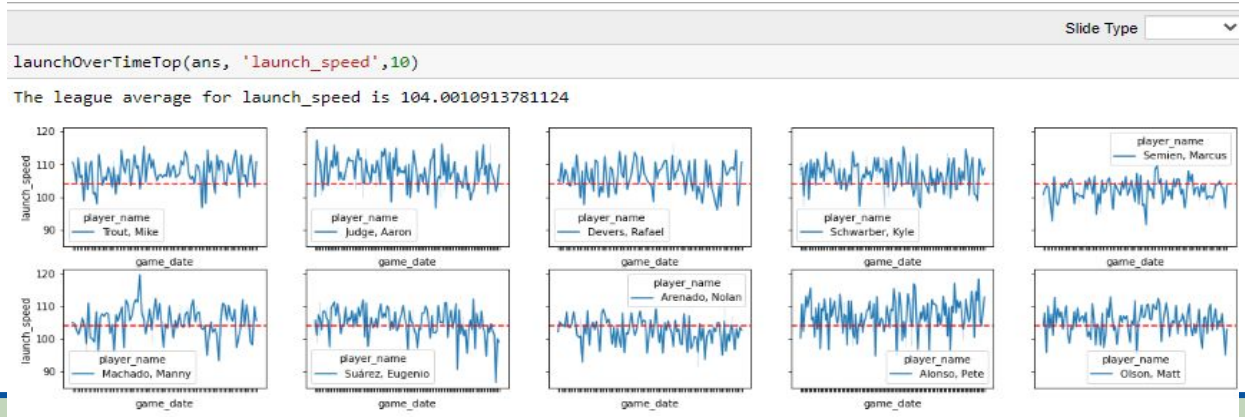
```
def y_coord(row):
    if row['zone'] in [1,2,3]:
        return 3
    elif row['zone'] in [4,5,6]:
        return 2
    elif row['zone'] in [7,8,9]:
        return 1


#Assigns new columns with x and y coordinates in order to graph the heatmap of homeruns
df['xcoord'] = df.apply(x_coord, axis = 1)
df['ycoord'] = df.apply(y_coord, axis = 1)

#Using seaborn, creates a histogram plot (heatmap) with a bar for the user to translate the color on the
ax = sns.histplot(data = df, x  = 'xcoord', y = 'ycoord', discrete = True, cmap ='Reds',cbar = True)
#Removes the left and bottom spine of the graph
sns.despine(left=True, bottom = True)
#Removes the x and y ticks from the graphs
ax.set(yticks=[], xticks = [])
ax.set_xlabel('Zone')
ax.set_ylabel("")
ax.set_title(f"Home Run Distribution for {playerName}")

#Count initializes in order to for the strike zone to be illustrated by the graph
count = 0

#Iterates through the cartesian plane (x and y axis to annotate the heatmap with the strikezones)

for y in reversed(range(1, 4)):
    for x in range(1,4):
        count += 1
        plt.text(x, y, count, horizontalalignment='center', size='medium', color='grey', weight='heavy')
plt.show()
```

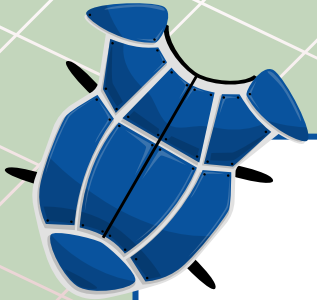
```
plotHeatMap(ans, 'Shohei')
```

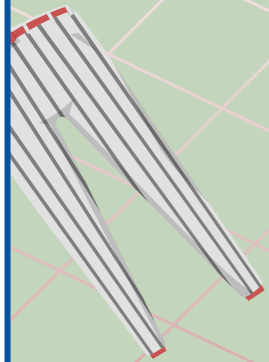
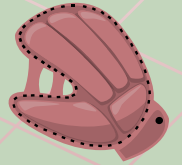


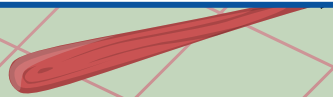
Top N Player Comparison

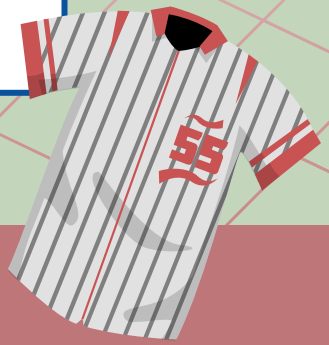
- This functions sorts through the top n players as specified by the function as judged by total home run count
- After, it plots their launch speed throughout the duration of the season to see if these two variables have a correlation
    - Since the graph is erratic, like the stock market graphs, these two variables do not have much of a correlation.









01

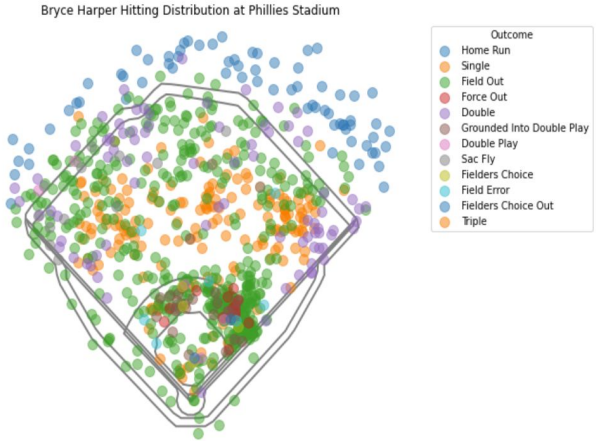
PyBaseball

Working with its functionality

PyBaseball

- Built-in Python package which allows for large baseball statistic downloads as the BaseballSavant website limits the amount of data requested
  - Scrapes multiple websites and stores the information about a player/team in dataframes for further analysis
- Can gather pitching statistics, hitting statistics, win and loss percentages, etc.

- **Note:** It may take long to download these datasets and points if the request is over a long period of time (normally long than three seasons)




02
Webscraping
BeautifulSoup + Pandas Implementation

BeautifulSoup

- BeautifulSoup is a python library which is helpful for reading HTML data and web scraping
  - Works with the requests library to establish a connection with the website to scrape the pages of information
- Finds all HTML tags on a page, which can be stored in lists or pandas dataframes
  - We are scraping the MLB database and the salary for each player

Popular Tags in HTML

Tags	Definition	Tips
<a href>-	link to another website	Format: <a href> title)
-	a division/section of a html site	
-	An inline container for inline elements on a site	
<class>-	An attribute of the inside of the div/span sections.	Multiple elements can have the same class name.
, <h#>, <p>	Lists, Headers, & Paragraphs in HTML (w text)	Header size range from h1-h6
Block Tag-	Tags that starts a new line & takes up the full width available	Sometimes contain inline tag (ex:<p>)
Inline Tag-	Tags that don't start a new line & don't take up full width	Inline tags can't have block tag (ex:)

Inspecting Elements on a Web Page

- There is built-in functionality in our computers that make web scraping in BeautifulSoup easy and effective: right click and inspect element
  - Displays the format of the website in its HTML language which is the language we will parse into an instance of the BeautifulSoup class
- https://www.spotrac.com/mlb/rankings/2020/salary/
- https://www.mlb.com/stats/home-runs/2020

spotrac

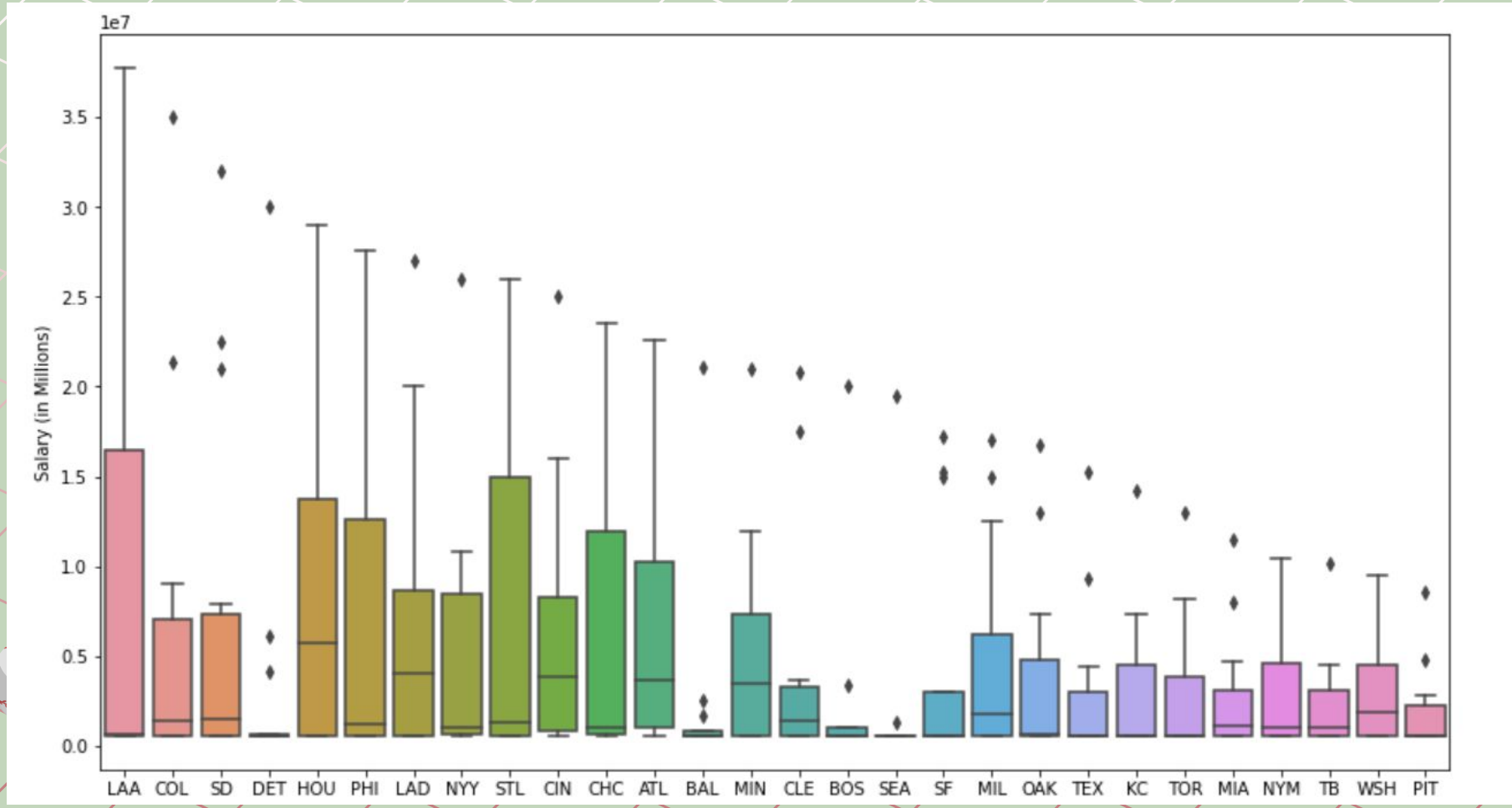
Data Cleaning

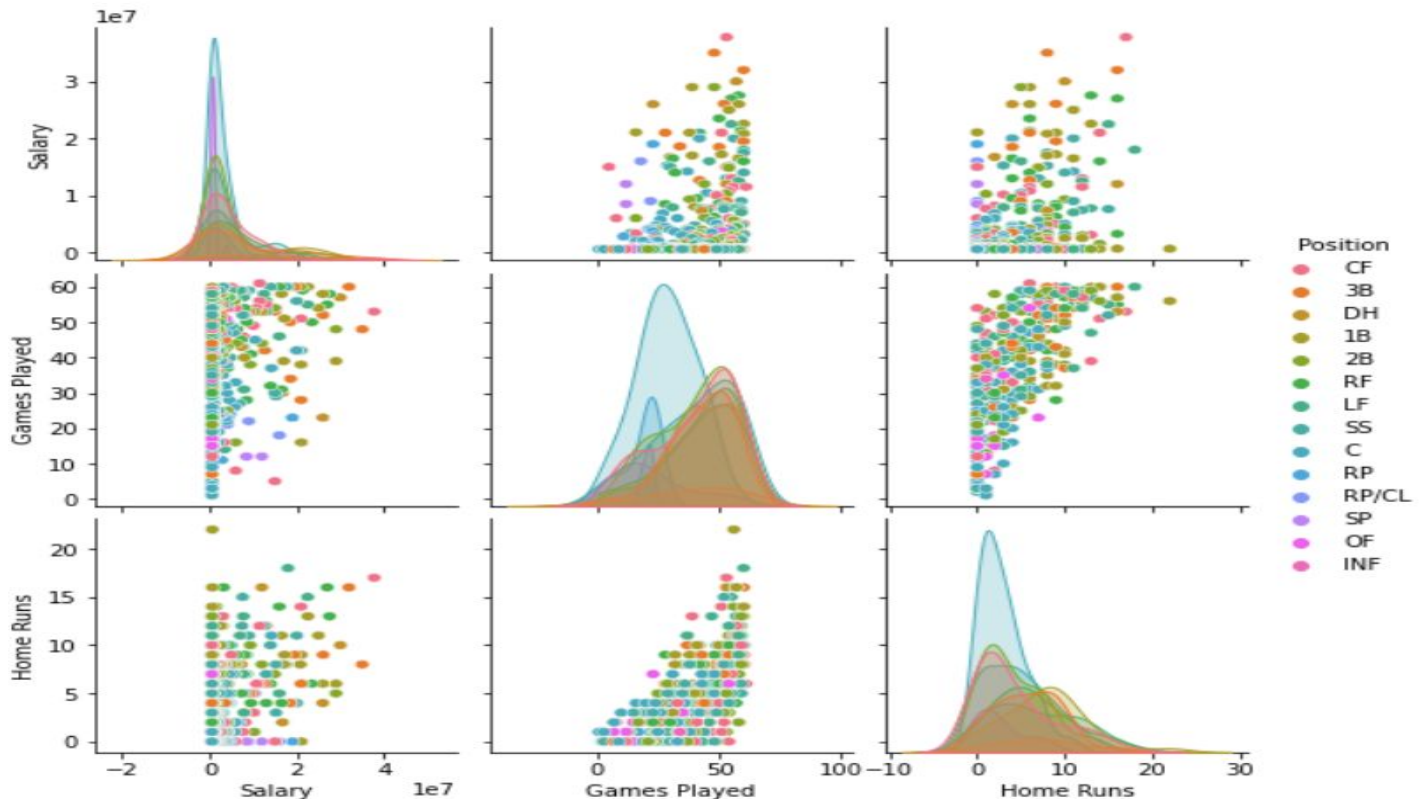
- Essential component of Data Science to properly format data in a manner in which performing calculations are easy and effective
- Process of rectifying incorrect data, deleting corrupted & duplicate data, reformatting data, and fixing/removing incomplete data.
- Regarding this dataset, it involves converting the salary from string to integer values, properly formatting the player names, and formatting the column names in a readable manner

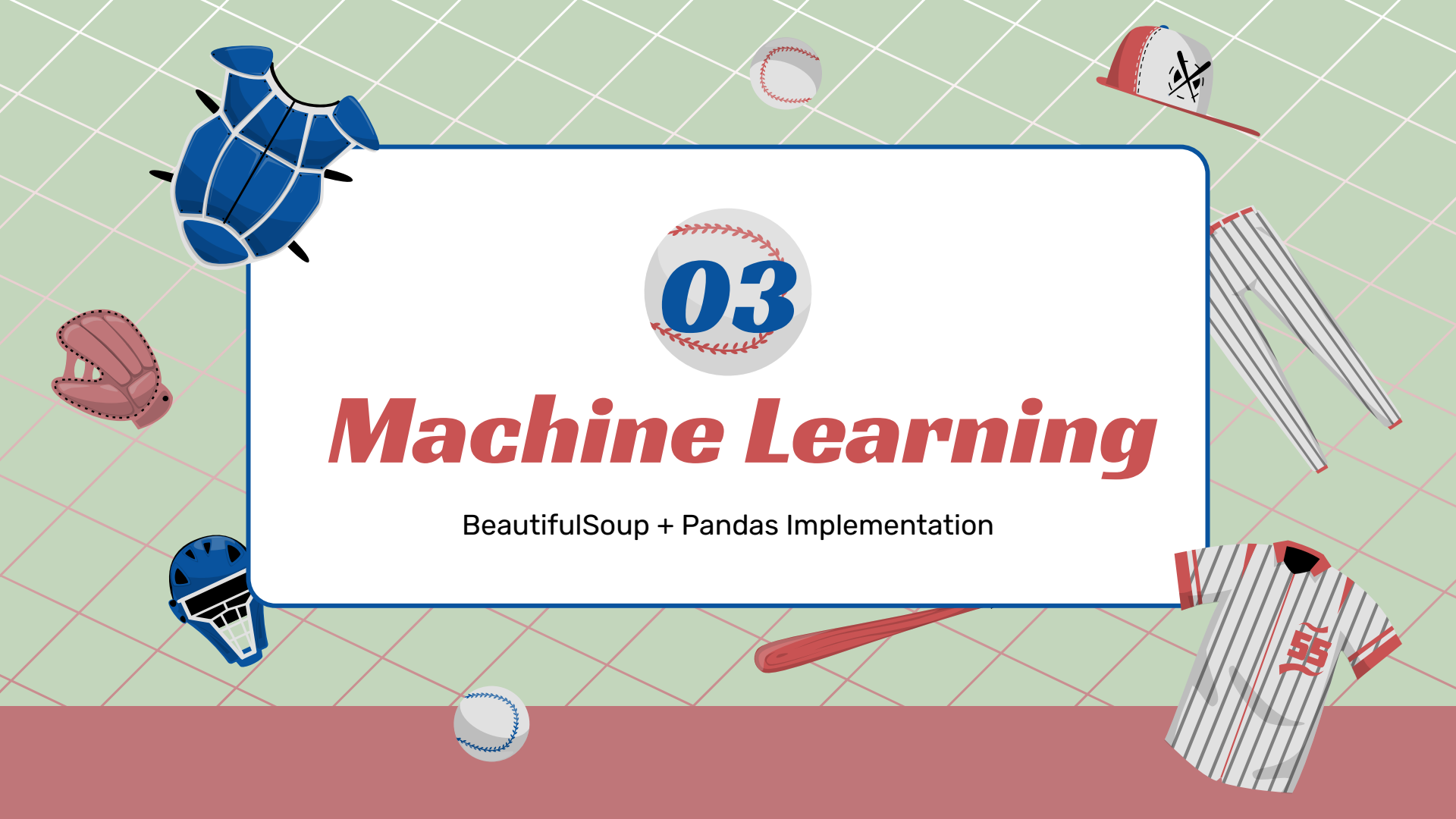
Web Scraping

```
#Since a lot of the page begins to load in as we scroll through the page, we need this to gather all of the information
response = requests.post(weblink, data={'ajax': 'true','mobile': 'false'}).content

soup = BeautifulSoup(response)

#Tr tags in html establish rows of a table. Therefore, we need to find these rows in the table to properly scrape the data
column_headers = soup.find_all('tr')[0]

#Goes through all of the header cells and formats them correctly to score them in the table
column_headers = [i.text.strip().title() for i in column_headers.find_all('th')]
column_headers = column_headers[1:]

#This accesses each block or row of the players in the website
baseballPlayers = soup.find_all('tr')[1:]

playerName = []
position = []
age = []
salary = []
team = []

#Iterate through all of the baseball players and store their information in the table
for player in baseballPlayers:
    #Accesses all h3 href attributes in order to properly find the player name
    playerName.append(player.h3.a.text)
    #Looks for the center small class which is where the position of the player is found
    position.append(player.find('td', class_ = 'center small').text.strip())
    #Finds the salary and gets rid of the dollar sign because we want to convert this to a numeric type later
    salary.append(player.find('span', class_ = 'info').text.strip().replace('$',""))
    #Looks for each div tag with the specific class and formats correctly
    team.append(player.find('div', class_ = 'rank-position').text.strip())
    #There are two types of center small classes, so we have to access the number attributes to gather the age
    for ages in player.find_all("td", {"class": "center small"}):
        plainText = ages.text.strip()

        #If its numeric, we know that it is for the age
        if(plainText.isnumeric()):
            age.append(plainText)

#Formats all of the data in the list into a pandas dataframe to gather all of the information properly
playerData = pd.DataFrame({"Player": playerName, 'Team': team, 'Age':age,"Position" : position, "Salary": salary})
playerData
```

```
#Want to scrape 25 pages from the 2020 season to compare players and salary
hitData = pd.DataFrame()

#Want to scrape all of the 25 pages from the MLB database
link = "https://www.mlb.com/stats/doubles/2020"
for i in range(2, 25):
    r = requests.get(link)
    #Keeps adding rows to the dataframe column with each page
    hitData = pd.concat([hitData, (pd.read_html(r.text, header = 0)[0])])
    link = f"https://www.mlb.com/stats/doubles/2020?page={i}"

#Resets the index to make accessing elements easier
hitData = hitData.reset_index()
hitData.drop('index', axis = 1, inplace = True)

#Renaming all of the columns to get rid of an errors that may have been produced with webscraping
column_headers = ['Player', 'Team', 'Games Played', 'At Bats', 'Runs', 'Hits', 'Doubles', 'Triples', 'Home Runs', 'RBIs',
                  'Walks','Strikeouts', 'Stolen Bases', 'Caught Stealing', 'Batting Average', 'On Base Percentage',
                  'Slugging Percentage', 'On Base Plus Slugging']

hitData.columns = column_headers

#Wants to split the player column with any capital letters that it may find
replaceNames = hitData['Player'].str.findall('[A-Z][^A-Z]*')

playerNames = []

#Appends each name in the list with the first and last name, assigning it to the new Player column
for names in replaceNames:
    playerNames.append(names[0] + " " + names[2])

hitData['Player'] = playerNames
hitData
```

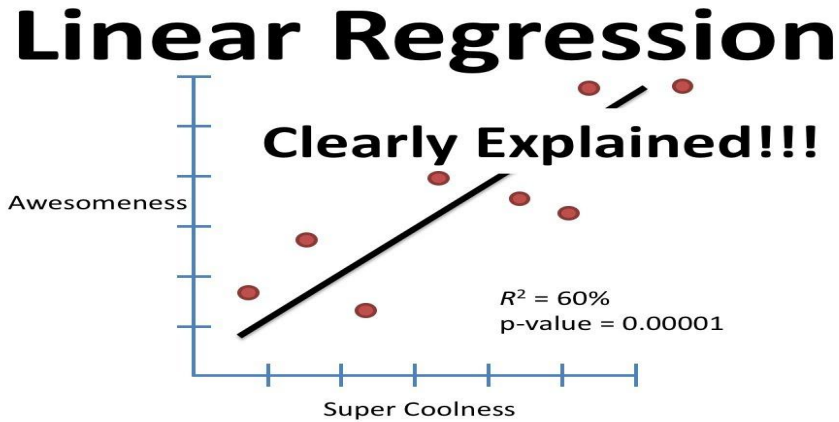

1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Salary (in Millions)
LAA
COL
SD
DET
HOU
PHI
LAD
NYY
STL
CIN
CHC
ATL
BAL
MIN
CLE
BOS
SEA
SF
MIL
OAK
TEX
KC
TOR
MIA
NYM
TB
WSH
PIT


1e7
Salary
Games Played
Home Runs
Position
CF
3B
DH
1B
2B
RF
LF
SS
C
RP
RP/CL
SP
OF
INF
1e7


03
Machine Learning
BeautifulSoup + Pandas Implementation

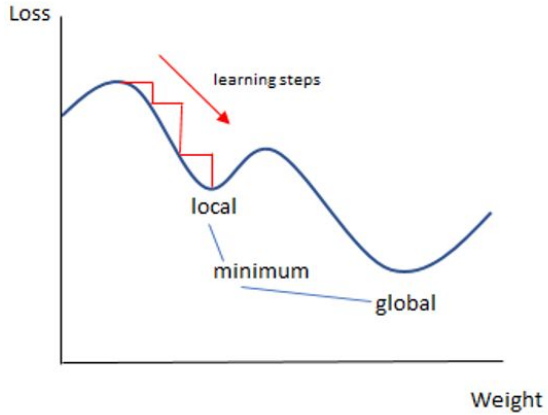
Linear Regression

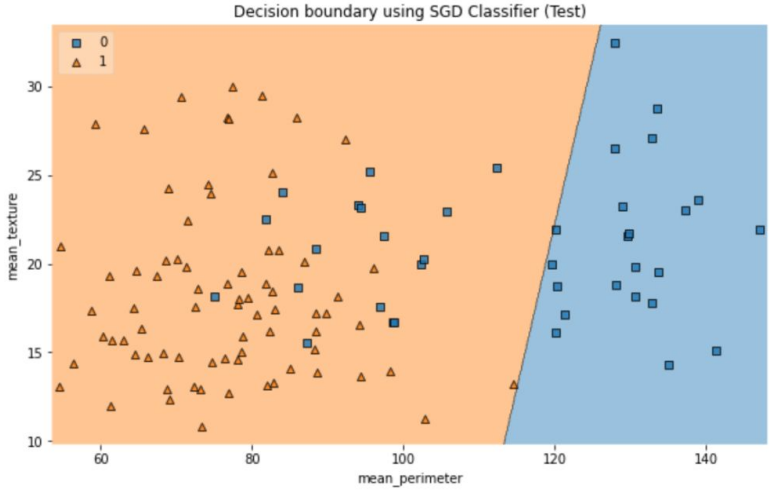
- Simple yet effective algorithm where we try to see if there is a correlation/relationship between the 2 variables
- We use the fit function on the xtrain & ytrain data. The model attempts to find characteristics between the two variables
- We use the trained algorithm to predict the test values & get a score



SGDClassifier

- Also known as Stochastic Gradient Descent Classifier
- Linear classifier which is optimized by the Stochastic Gradient
- Setting loss = ‘log’ simulates an optimized Logistic Regression model
- While this is not an accurate model, it allows for a general prediction of salary when given multiple inputs



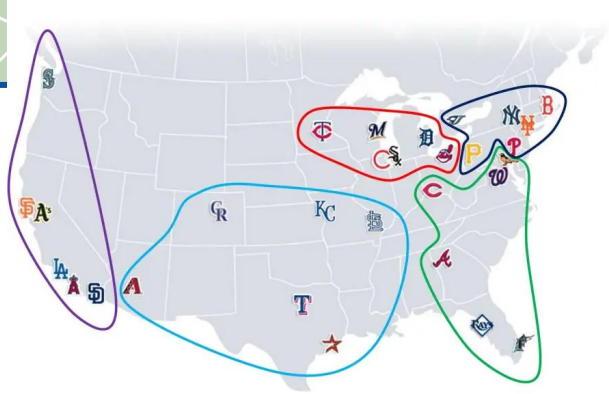


How to Approach Machine Learning

1. Data Collection
2. Data Preparation
3. Choosing a Model
4. Feature Selection

1. Training the Model
2. Evaluate the Model
3. Hyperparameter Tuning
4. Make Predictions

Data Collection



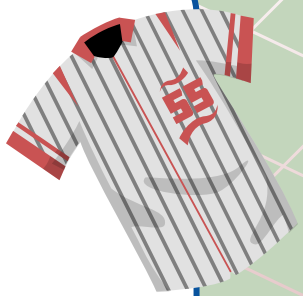
We decided to try to predict what division a given MLB team is in given some data about their performance over a year.

We just want to make sure that the data we collect is made accurately, as well as having lots of it. Also, we have to consider what features we want to include in the model that predict our target variable. Obviously, features like temperature and location would give away the true answer quickly, but we want to see if other factors in a game can be used to predict a team's division.

```
# Let's count how many teams we have for each division played over all years

print(df_teams_no_NA[df_teams_no_NA['divID'] == 'W'].shape[0])
print(df_teams_no_NA[df_teams_no_NA['divID'] == 'C'].shape[0])
print(df_teams_no_NA[df_teams_no_NA['divID'] == 'E'].shape[0])
```

```
575
295
598
```

Data Preparation



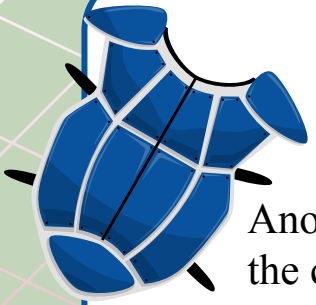
We now want to clean the data. This involves dealing with missing values, removing duplicates, fixing errors, and more.

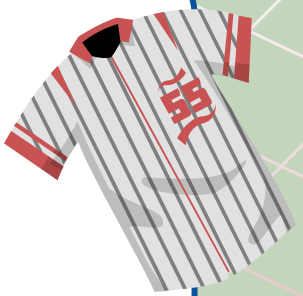
We can also visualize the data to find relationships between variables.

By exploring the feature data, we can remove those that are irrelevant as well as ones we have limited data on.

```
# (remove NaN values)

# Remove data where the division ID does not exist

df_teams_no_NA = df_teams_red[df_teams_red['divID'].isna() == False].copy()
df_teams_no_NA
```

Data Preparation





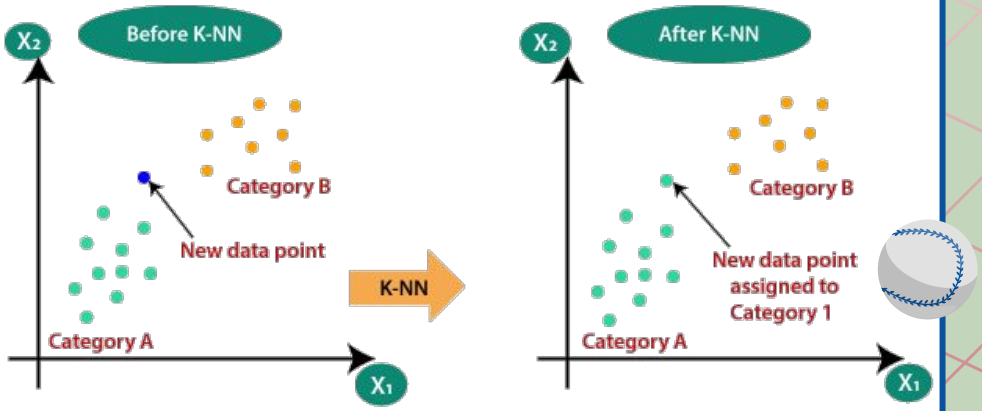
Another step is determining what variables are categorical and numerical. In the case we have a categorical variable, we will encode that into numerical values (for example in the binary case, ‘guilty’: 0, ‘innocent’: 1).

```
from sklearn.preprocessing import OneHotEncoder

# Convert column type to category
df_teams_no_NA['WSWin'] = df_teams_no_NA['WSWin'].astype('category')
enc = OneHotEncoder(drop = 'first')
enc_data = enc.fit_transform(df_teams_no_NA['WSWin'].values.reshape(-1,1)).toarray()
df_teams_no_NA['WSWin_enc'] = enc_data
df_WSWin_orig = df_teams_no_NA['WSWin'].copy()
df_teams_no_NA.drop(['WSWin'], axis = 1, inplace=True)

df_teams_no_NA['divID'] = df_teams_no_NA['divID'].astype('category')
```

Choosing a Model

Because we are classifying division type for data that include the true division, we will choose a supervised classification model.

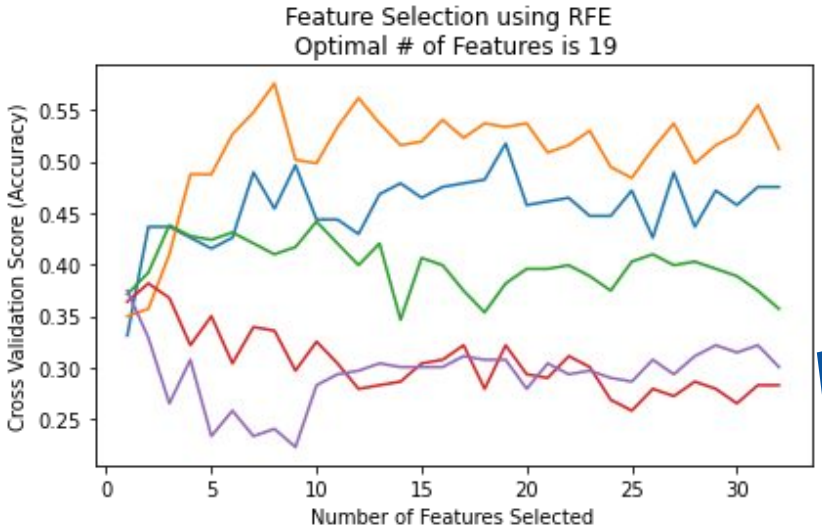
In this case we will use the **k-Nearest Neighbor algorithm**. The training time of this is short, however irrelevant data can hurt its accuracy. Also it is better suited for lower dimensional data. We can reduce the dimensionality of the input features by using feature selection.



Feature Selection

We decided to try to predict what division a given MLB team is in given some data about their performance over a year.

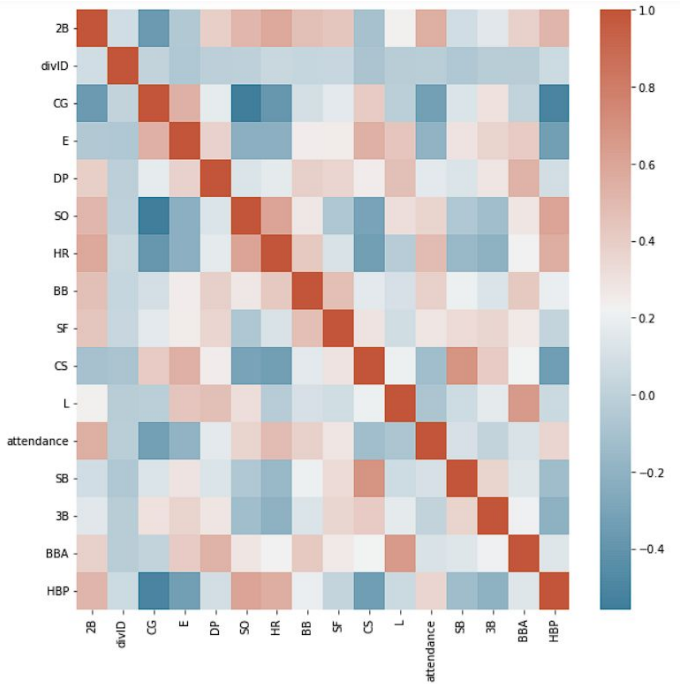
We just want to make sure that the data we collect is made accurately, as well as having lots of it. Also, we have to consider what features we want to include in the model that predict our target variable. Obviously, features like temperature and location would give away the true answer quickly, but we want to see if other factors in a game can be used to predict a team's division.



Feature Selection

We employed various methods for selecting features we want to keep for the model:

1. Correlation matrix
    a. Avoid features with high multicollinearity (when independent variables are highly correlated with each other) which can cause overfitting
    b. Multicollinearity makes it it difficult to identify how important
    c. Eliminate redundancy (two features providing the same information about the response variable, i.e. Games played home & Games played)



Feature Selection

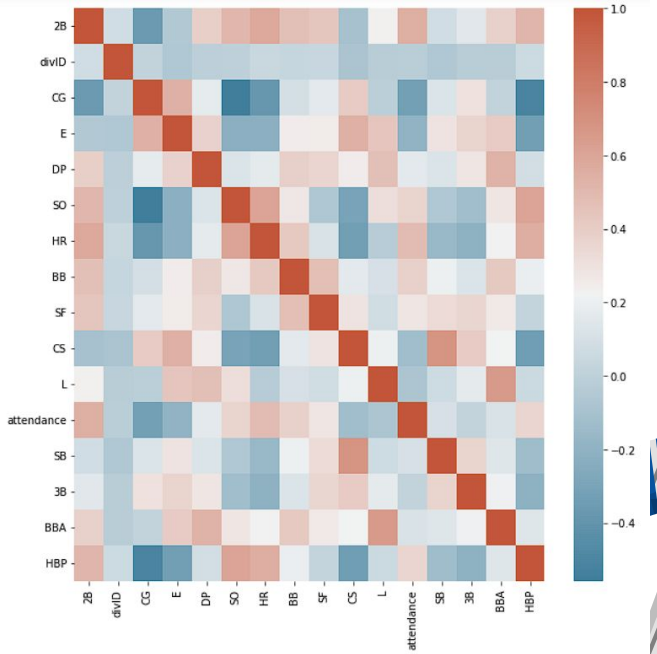
We employed various methods for selecting features we want to keep for the model:

1. Correlation matrix

- Using Pandas dataframe.corr() to find the **Pearson pairwise correlation coefficient**
- Ignores non-numeric data and NaN values

$$r = \frac{\sum (x_i - x)(y_i - y)}{\sqrt{\sum (x_i - x)^2 \sum (y_i - y)^2}}$$

- r > .5 indicates strong positive correlation
- r < -.5 indicates strong negative correlation
- We will remove input features with high |r| values.

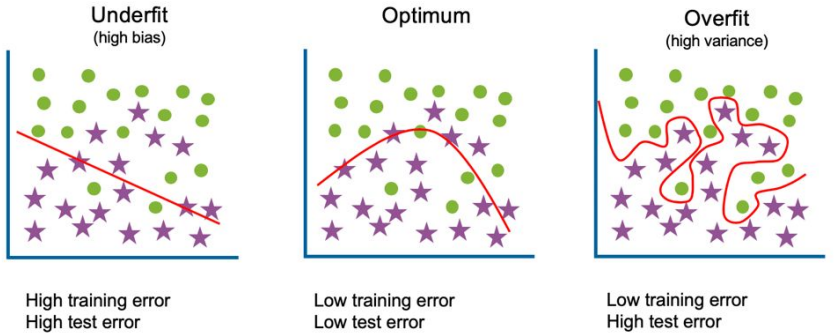


Training the Model

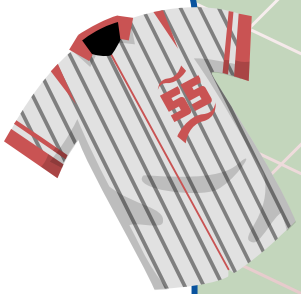
When training the model, we must try our best to prevent overfitting, when the model performs well on training data but not test data.

We want the model to make a prediction off the test data as correctly as possible.

Before training, we should check the distribution of our features, as well as whether or not scaling is necessary. kNN calculates distances between data, so we do not want it to be affected by the different magnitudes of the variables.



Training the Model



Luckily, standardization does not transform the underlying distribution, so we do not have to assume that our features are normally distributed before standardizing.

```
# Standardize data first

scale = StandardScaler()
X_scaled = scale.fit_transform(X)
```

Most of the features are normally distributed, so their resulting distributions will follow the standard normal distribution: $Z = \frac{X - \mu}{\sigma} \sim N(0, 1)$

We also split up the training and testing data randomly to avoid overfitting the model.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.2,
                                                    random_state = 42)
```

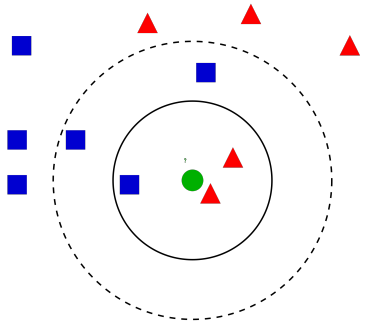
Training the Model

kNN Algorithm

1. Initialize *k* (# of neighbors to choose from)
2. Calculate distance between test point and all training points using Euclidean distance
3. Sort distances in increasing order and choose the first *k* points
4. Return the mode of the *k* labels

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

```
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(X_train, y_train)
```



Evaluate the Model

- Test the model on previously unseen data
  - Score the model using the 20% testing dataset
- Measure the performance of the *k*NN model

```
print(knn.score(X_test, y_test))
```

```
0.5492957746478874
```

Accuracy (binary classification)
= (TP + TN) /
(TP + TN + FP + FN)

$$Accuracy = \frac{T_W + T_C + T_E}{T_W + T_C + T_E + F_W + F_C + F_W}$$

Let's try optimizing first before evaluating the classification model further

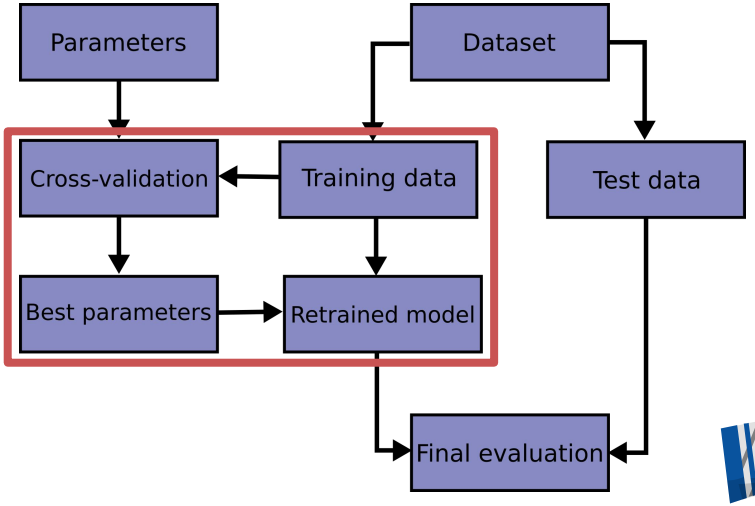
Hyperparameter Tuning

Hyperparameters are special parameters that are used to tweak the model.

Advantage of *k*NN:

- low # of hyperparameters

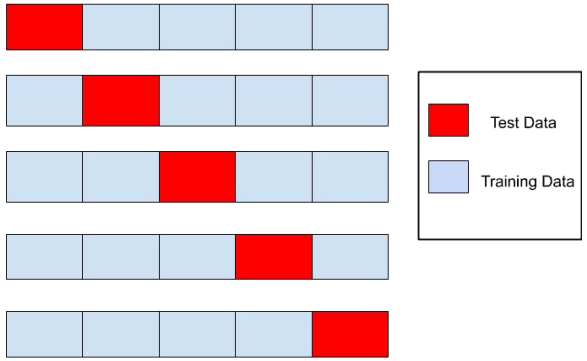
Interested in *k* (number of neighbors)



While optimizing *k*, we will use **k-Fold Cross-Validation**. This splits up the data randomly into k groups, in this case we will set k to 5. The dataset will therefore be split up into 5 groups, trained and tested separately. We will then compare the mean scores.

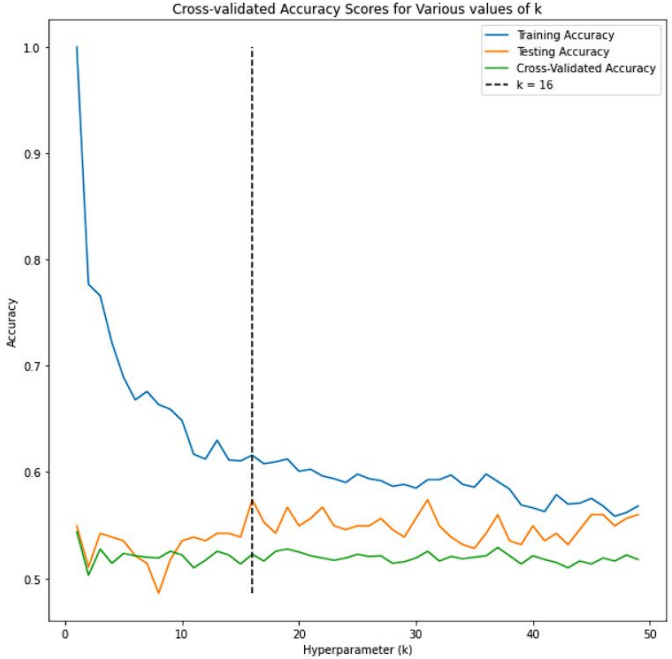
Hyperparameter Tuning

```
neighbors = np.arange(1, 50)
train_accs = np.empty(len(neighbors))
test_accs = np.empty(len(neighbors))
cross_val_accs = np.empty(len(neighbors))

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)
    train_accs[k - 1] = knn.score(X_train, y_train)
    test_accs[k - 1] = knn.score(X_test, y_test)

    knn_cv = KNeighborsClassifier(n_neighbors = k)
    cv_scores = cross_val_score(knn_cv,
                                X_scaled,
                                y,
                                cv = ms.KFold(shuffle = True,
                                              random_state = 42
                                             )
                               )
    cross_val_accs[k - 1] = np.mean(cv_scores)
```



Hyperparameter Tuning

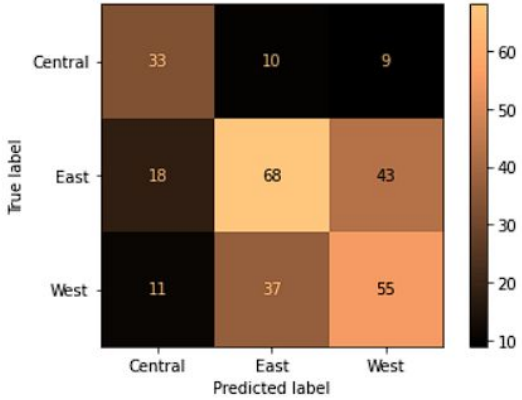
```
# Plotting accuracies

plt.figure(figsize=(10,10))
plt.plot(neighbors, train_accs,
         label = 'Training Accuracy')
plt.plot(neighbors, test_accs,
         label = 'Testing Accuracy')
plt.plot(neighbors, cross_val_accs,
         label = 'Cross-Validated Accuracy')
plt.vlines(x = list(test_accs).index(max(test_accs)) + 1,
           ymin = min(np.hstack((train_accs,
                                 test_accs,
                                 cross_val_accs))),
           ymax = max(np.hstack((train_accs,
                                 test_accs,
                                 cross_val_accs))),
           colors = 'black',
           linestyle = 'dashed',
           label = f'k = {list(test_accs).index(max(test_accs)) + 1}')
```

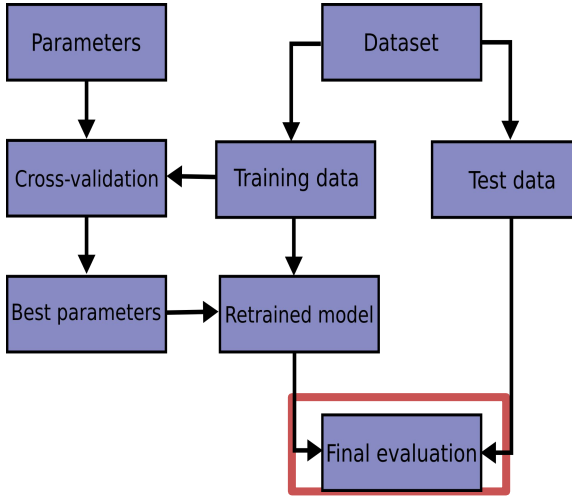


Optimal *k* value was found to be 16. Let's now re-evaluate the model.

Re-Evaluate the Model

```
labels = np.array(['Central','East','West'])
conf_mat_disp = ConfusionMatrixDisplay.from_estimator(knn_temp,
                                                      X_test,
                                                      y_test,
                                                      display_labels = labels,
                                                      cmap = 'copper')
plt.show()
print(classification_report(y_test, knn_temp.predict(X_test)))
```

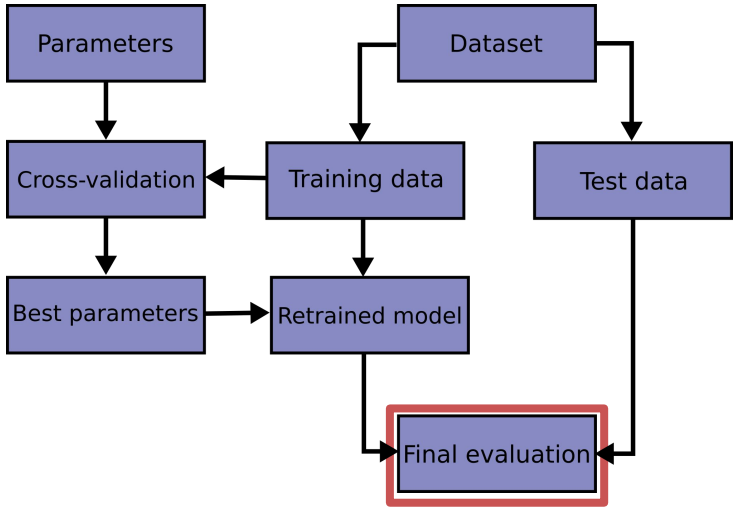




Re-Evaluate the Model

	precision	recall	f1-score	support
C	0.53	0.63	0.58	52
E	0.59	0.53	0.56	129
W	0.51	0.53	0.52	103
accuracy			0.55	284
macro avg	0.55	0.57	0.55	284
weighted avg	0.55	0.55	0.55	284

Off of the f1-scores, we can see that the classifier predicts each division with roughly equal accuracy and predicts Central division teams the best.



Make Predictions

Predict off yearly statistics (test data)

```
print(knn_temp.predict(X_test))
print(extracted_features)
```

```
['C' 'E' 'E' 'W' 'W' 'E' 'W' 'W' 'C' 'E' 'W' 'E' 'C' 'C' 'E' 'E' 'W' 'E'
 'E' 'C' 'E' 'C' 'E' 'W' 'W' 'C' 'C' 'W' 'C' 'C' 'W' 'W' 'E' 'W' 'W' 'C'
 'E' 'W' 'E' 'E' 'W' 'W' 'C' 'E' 'W' 'E' 'E' 'C' 'C' 'W' 'W' 'C' 'E' 'E'
 'E' 'E' 'E' 'E' 'E' 'C' 'W' 'W' 'W' 'E' 'C' 'E' 'W' 'C' 'W' 'E' 'W' 'W'
 'E' 'W' 'W' 'W' 'E' 'E' 'C' 'C' 'E' 'E' 'W' 'W' 'E' 'W' 'W' 'E' 'E' 'C'
 'E' 'C' 'W' 'E' 'C' 'W' 'W' 'W' 'W' 'E' 'W' 'E' 'W' 'W' 'W' 'W' 'C' 'E'
 'E' 'E' 'C' 'W' 'C' 'W' 'W' 'E' 'W' 'W' 'C' 'E' 'W' 'C' 'C' 'C' 'C' 'E'
 'C' 'E' 'C' 'C' 'E' 'E' 'W' 'W' 'W' 'W' 'C' 'W' 'C' 'E' 'C' 'W' 'W' 'W'
 'E' 'C' 'W' 'E' 'E' 'E' 'E' 'C' 'E' 'C' 'E' 'C' 'W' 'E' 'W' 'E' 'C' 'W'
 'W' 'C' 'C' 'E' 'C' 'W' 'E' 'C' 'C' 'E' 'C' 'E' 'C' 'W' 'E' 'W' 'E' 'W'
 'W' 'E' 'W' 'E' 'W' 'C' 'W' 'W' 'E' 'E' 'W' 'C' 'C' 'E' 'E' 'W' 'W' 'W'
 'E' 'E' 'E' 'C' 'W' 'C' 'W' 'W' 'E' 'W' 'E' 'E' 'C' 'E' 'E' 'W' 'E' 'W'
 'W' 'W' 'E' 'E' 'W' 'E' 'W' 'C' 'E' 'W' 'C' 'E' 'C' 'C' 'E' 'C' 'C' 'C'
 'C' 'C' 'C' 'E' 'E' 'E' 'E' 'E' 'W' 'E' 'E' 'W' 'E' 'E' 'W' 'W' 'W' 'C'
 'C' 'E' 'E' 'E' 'E' 'W' 'E' 'W' 'W' 'E' 'E' 'W' 'W' 'E' 'E' 'W' 'C' 'W'
 'E' 'C' 'C' 'W' 'E' 'W' 'W' 'C' 'W' 'C' 'W' 'E' 'E' 'C']
['2B', 'divID', 'CG', 'E', 'SO', 'HR', 'BB', 'SF', 'CS', 'L', 'attendance', 'SB', '3B', 'BBA', 'HBP']
```

2B - Doubles
CG - Complete games
E - Errors
SO - Strikeouts by batters
HR - Homeruns by batters
BB - Walks by batters
SF - Sacrifice flies
CS - Caught stealing
L - Losses
Attendance - Home attendance total
SB - Stolen bases
3B - Triples
BBA - Walks allowed
HBP - Batters hit by pitch
divID - Team's division

Make Predictions

Predict off yearly statistics (let's use data from the Florida Marlins (East Division) in the 2019 season)

```
print(f"Predicted Division: {knn_temp.predict(np.array(X_temp_standardized))}")
print(f"True Division: {true_div}")
```

```
Please enter a year the team played in: 2019
Please enter the teamID of the team: MIA
Predicted Division: ['E']
True Division: E
```

Make Predictions

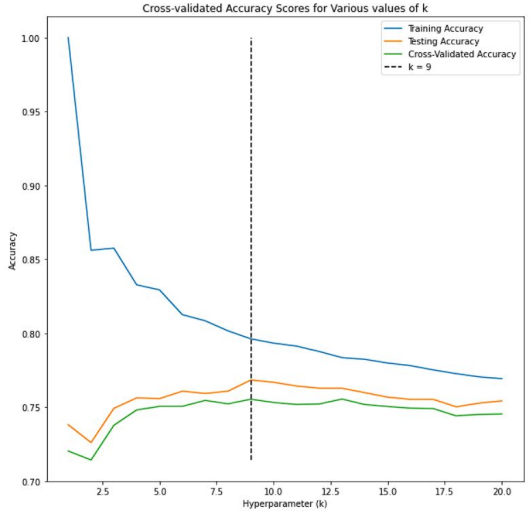
Predict off yearly statistics (changing individual features slightly)

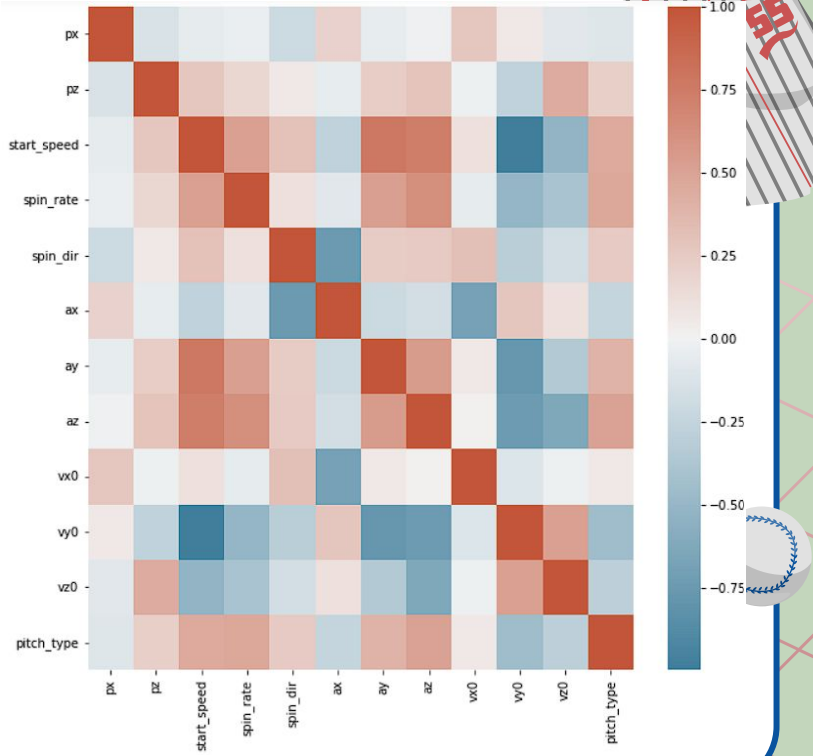
```
print(X_temp)
X_temp_new = np.array([[280,4,100,1300,100,300,20,50,100,2800000,50,10,700,100]])
print(X_temp_new)
X_temp_new_standardized = []
count = 0
for c in X_temp.columns:
    X_temp_new_standardized.append((X_temp_new[0, count] - X_temp_means[count]) / X_temp_stds[count])
    count += 1
print(X_temp_new_standardized)
print(knn_temp.predict(np.array([X_temp_new_standardized])))
```

```
       2B  CG   E      SO   HR     BB    SF    CS    L  attendance    SB  3B  \
2909  265   2  94  1469.0  146  395.0  33.0  30.0  105    811302.0  55.0  18

      BBA   HBP
2909  615  73.0
[[    280       4     100    1300     100     300      20      50     100
  2800000      50      10     700     100]]
[0.44352433615886866, -0.7420935008036289, -0.5121244913738869, 1.331524628723477, -1.138321072637420
2, -2.5311270439814297, -2.382261264054009, 0.22478655631528408, 1.5000338780595082, 0.93991049638392
91, -1.26071524086735, -2.0266773648899834, 2.16438255907792, 3.0722241899729688]
['C']
```

Predicting Pitch Type

Using the method we just described for predicting the division type, let's now try to predict the pitch type given pitch data.

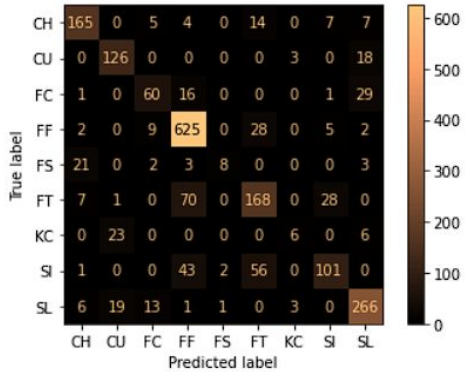




Predicting Pitch Type

After following the steps outlined previously, we get the following results.

```
conf_mat_disp = ConfusionMatrixDisplay.from_estimator(knn_temp,
                                                      X_test,
                                                      y_test,
                                                      cmap = 'copper')
plt.show()
print(classification_report(y_test, knn_temp.predict(X_test)))
```



	precision	recall	f1-score	support
CH	0.81	0.82	0.81	202
CU	0.75	0.86	0.80	147
FC	0.67	0.56	0.61	107
FF	0.82	0.93	0.87	671
FS	0.73	0.22	0.33	37
FT	0.63	0.61	0.62	274
KC	0.50	0.17	0.26	35
SI	0.71	0.50	0.59	203
SL	0.80	0.86	0.83	309
accuracy			0.77	1985
macro avg	0.71	0.61	0.64	1985
weighted avg	0.76	0.77	0.76	1985

Conclusions

1. Features
   a. Analyze distribution
   b. Try using different features
   c. Significant features
   d. Dimensionality Reduction with PCA
2. Evaluating
   a. Performance metrics for imbalanced classification
   b. ROC curve
3. Predicting
   a. Probability-based prediction
   b. Graph predictions by feature