Assignment 1: Data Analysis - Kershaw Pitch Data

Group members :

Maya Elshweikhy - 900204233

Youssef Aboukhatwa - 900211153

Ali Baligh - 900211718

Contents

Kershaw Pitch Data Description

In this python notebook, we will be representing a description of the variables in the data Kershaw by using statistics, visuals, and graphs to analyze the data. This data is Pitch-by-pitch data for baseball pitcher Clayton Kershaw in the 2013 season. Dataset includes information for 3,402 individual pitches thrown by Los Angeles Dodger baseball pitcher Clayton Kershaw during the 2013 regular season when he won the Cy Young award as the best pitcher in the National League. Many variables are measured using Major League Baseball's PITCHf/x system that uses camera systems in each ballpark to track characteristics of each pitch thrown. It is a data frame with 3402 observations on 24 variables.

Variables Description

- BatterNumber: Number of batters faced so far that game
- Outcome: One of 14 possible results for a pitch (e.g. Ball, Ball In Dirt, Called Strike, ..., Swinging Strike (Blocked))
- Class: One of three classifications (B=ball, S=strike, or X=in play)
- Result: From pitcher's perspective (Neg=ball or hit, Pos=strike or out)

- Swing: Did the batter swing at the pitch? (No or Yes)
- Time: Date and time of the pitch (format yyyy-mm-ddThh:mm:ssZ)
- StartSpeed: Speed leaving the pitcher's hand (in mph)
- EndSpeed: Speed crossing home plate (in mph)
- HDev: Horizontal movement (inches)
- VDev: Vertical movement (inches)
- HPos: Horizontal position at home plate (inches from center, positive is catcher's right)
- VPos: Vertical position at home plate (inches above the ground)
- PitchType: Code for pitch type (CH=changeup, CU=curve, FF=fastball, or SL=slider)
- Zone: 1-9 in theoretical strike zone (upper left to lower right), 11-14 are out of strike zone
- Nasty: A measure on a 0-100 scale of difficulty of the pitch to hit (100 is most difficult)
- Count: Ball strike count (0-0, 0-1, 0-2, 1-1, 1-2, 2-1, 2-2, 3-1, or 3-2)
- BallCount: Number of balls before the pitch (0, 1, 2, or 3)
- StrikeCount: Number of strikes before the pitch (0, 1, or 2)
- Inning: Inning of the game
- InningSide: Portion of the inning (bottom= pitcher at home or top=pitcher away)
- Outs: Number of outs when the pitch is thrown
- BatterHand: Batter's stance (L=left or R=right)
- ABEvent: Result of the at bat (several possibilities)
- Batter: Name of the batter faced

Source

- Data scraped from the MLB GameDay website (http://gd2.mlb.com/components/game/mlb/ (http://gd2.mlb.com/components/game/mlb/)) using pitchRx

Part 1: Numerical Summaries

In [1]:

```
import pandas as pd
```

In [2]:

```
df=pd.read_csv('Kershaw.csv')
```

In [4]:

```
df
```

Out[4]:

	Unnamed: 0	BatterNumber	Outcome	Class	Result	Swing	Time	StartSpeed	En
0	1	1	Called Strike	S	Pos	No	2013-04-01T20:15:04Z	92.7	
1	2	1	In play, out(s)	X	Pos	Yes	2013-04-01T20:15:20Z	73.1	
2	3	2	Ball	B	Neg	No	2013-04-01T20:16:01Z	92.4	
3	4	2	Called Strike	S	Pos	No	2013-04-01T20:16:14Z	92.6	
4	5	2	Ball	B	Neg	No	2013-04-01T20:16:46Z	72.7	
...	...	...	...	...	...	...	...	...	
3397	3398	51	Ball	B	Neg	No	2013-09-28T04:00:26Z	93.1	
3398	3399	51	Ball	B	Neg	No	2013-09-28T04:00:49Z	94.0	
3399	3400	51	Called Strike	S	Pos	No	2013-09-28T04:01:15Z	82.7	
3400	3401	51	Ball In Dirt	B	Neg	No	2013-09-28T04:01:43Z	86.4	
3401	3402	51	In play, out(s)	X	Pos	Yes	2013-09-28T04:02:15Z	92.9	

3402 rows × 25 columns

In [5]:

```
print("List of Columns:\n", df.columns)
```

```
List of Columns:
 Index(['Unnamed: 0', 'BatterNumber', 'Outcome', 'Class', 'Result',
'Swing',
       'Time', 'StartSpeed', 'EndSpeed', 'HDev', 'VDev', 'HPos', 'VPo
s',
       'PitchType', 'Zone', 'Nasty', 'Count', 'BallCount', 'StrikeCou
nt',
       'Inning', 'InningSide', 'Outs', 'BatterHand', 'ABEvent', 'Batt
er'],
      dtype='object')
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3402 entries, 0 to 3401
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     3402 non-null   int64  
 1   BatterNumber   3402 non-null   int64  
 2   Outcome        3402 non-null   object 
 3   Class          3402 non-null   object 
 4   Result         3402 non-null   object 
 5   Swing          3402 non-null   object 
 6   Time           3402 non-null   object 
 7   StartSpeed     3402 non-null   float64
 8   EndSpeed       3402 non-null   float64
 9   HDev           3402 non-null   float64
 10  VDev           3402 non-null   float64
 11  HPos           3402 non-null   float64
 12  VPos           3402 non-null   float64
 13  PitchType      3402 non-null   object 
 14  Zone           3402 non-null   int64  
 15  Nasty          3402 non-null   int64  
 16  Count          3402 non-null   object 
 17  BallCount      3402 non-null   int64  
 18  StrikeCount    3402 non-null   int64  
 19  Inning         3402 non-null   int64  
 20  InningSide     3402 non-null   object 
 21  Outs           3402 non-null   int64  
 22  BatterHand     3402 non-null   object 
 23  ABEvent        3402 non-null   object 
 24  Batter         3402 non-null   object 
dtypes: float64(6), int64(8), object(11)
memory usage: 664.6+ KB
```

For continuous variables, provide the mean, median, quartiles, variance, and standard deviation

In [7]:

```
df.describe()
```

Out[7]:

	Unnamed: 0	BatterNumber	StartSpeed	EndSpeed	HDev	VDev	
count	3402.000000	3402.000000	3402.000000	3402.000000	3402.000000	3402.000000	3402.(
mean	1701.500000	29.277190	88.170723	81.110494	-0.457128	6.326934	-2.2
std	982.217135	17.592011	6.484404	5.779163	2.749855	8.049808	8.3
min	1.000000	1.000000	69.500000	63.600000	-8.760000	-17.170000	-31.4
25%	851.250000	14.000000	85.000000	78.500000	-2.270000	1.140000	-8.(
50%	1701.500000	28.000000	91.700000	83.900000	-0.290000	10.320000	-2.7
75%	2551.750000	44.000000	92.800000	85.200000	1.180000	12.590000	3.1
max	3402.000000	74.000000	95.600000	88.100000	11.330000	17.850000	29.8

- **Variable 1 "StartSpeed"**

In [8]:

```
df.StartSpeed.mean()
```

Out[8]:

```
88.17072310405652
```

In [9]:

```
df.StartSpeed.median()
```

Out[9]:

```
91.7
```

In [10]:

```
df.StartSpeed.quantile(.5)
```

Out[10]:

```
91.7
```

In [11]:

```
df.StartSpeed.quantile(.25)
```

Out[11]:

```
85.0
```

In [10]:

```
df.StartSpeed.quantile(.75)
```

Out[10]:

```
92.8
```

In [11]:

```
df.StartSpeed.quantile([0,.25,.5,.75,1])
```

Out[11]:

```
0.00    69.5
0.25    85.0
0.50    91.7
0.75    92.8
1.00    95.6
Name: StartSpeed, dtype: float64
```

In [12]:

```
df.StartSpeed.std()
```

Out[12]:

```
6.484404288530577
```

In [13]:

```
df.StartSpeed.var()
```

Out[13]:

```
42.04749897711373
```

- **Variable 2 "EndSpeed"**

In [14]:

```
df.EndSpeed.mean()
```

Out[14]:

```
81.11049382716041
```

In [15]:

```
df.EndSpeed.median()
```

Out[15]:

```
83.9
```

In [16]:

```
df.EndSpeed.quantile([0,.25,.5,.75,1])
```

Out[16]:

```
0.00    63.6
0.25    78.5
0.50    83.9
0.75    85.2
1.00    88.1
Name: EndSpeed, dtype: float64
```

In [17]:

```
df.EndSpeed.std()
```

Out[17]:

```
5.779163297926592
```

In [18]:

```
df.EndSpeed.var()
```

Out[18]:

```
33.39872842410177
```

- **Variable 3 "HDev"**

In [19]:

```
df.HDev.mean()
```

Out[19]:

```
-0.4571281599059383
```

In [20]:

```
df.HDev.median()
```

Out[20]:

```
-0.29
```

In [21]:

```
df.HDev.quantile([0,.25,.5,.75,1])
```

Out[21]:

```
0.00    -8.76
0.25    -2.27
0.50    -0.29
0.75     1.18
1.00    11.33
Name: HDev, dtype: float64
```

In [22]:

```
df.HDev.std()
```

Out[22]:

```
2.749854559898511
```

In [23]:

```
df.HDev.var()
```

Out[23]:

```
7.561700100594634
```

- **Variable 4 "VDev"**

In [24]:

```
df.VDev.mean()
```

Out[24]:

```
6.326934156378598
```

In [25]:

```
df.VDev.median()
```

Out[25]:

```
10.32
```

In [26]:

```
df.VDev.quantile([0,.25,.5,.75,1])
```

Out[26]:

```
0.00   -17.17
0.25     1.14
0.50    10.32
0.75    12.59
1.00    17.85
Name: VDev, dtype: float64
```

In [27]:

```
df.VDev.std()
```

Out[27]:

```
8.049808472453151
```

In [28]:

```
df.VDev.var()
```

Out[28]:

```
64.79941644317853
```

- **Variable 5 "HPos"**

In [29]:

```
df.HPos.mean()
```

Out[29]:

```
-2.2548959435626155
```

In [30]:

```
df.HPos.median()
```

Out[30]:

```
-2.76
```

In [31]:

```
df.HPos.quantile([0,.25,.5,.75,1])
```

Out[31]:

```
0.00   -31.428
0.25    -8.064
0.50    -2.760
0.75     3.114
1.00    29.880
Name: HPos, dtype: float64
```

In [32]:

```
df.HPos.std()
```

Out[32]:

```
8.397873079412046
```

In [33]:

```
df.HPos.var()
```

Out[33]:

```
70.52427225791355
```

Part 2: Correlations

Correlations between variables table

In [55]:

```
x=df.corr()
x
```

Out[55]:

atterNumber	StartSpeed	EndSpeed	HDev	VDev	HPos	VPos	Zone	Na
0.026747	-0.007731	0.012000	0.055752	-0.058277	0.002985	-0.068199	0.011496	-0.008
1.000000	-0.115421	-0.108019	-0.030773	-0.087906	-0.003839	-0.033244	0.021421	0.004
-0.115421	1.000000	0.991489	0.508977	0.828273	-0.114238	0.198838	-0.133523	0.328
-0.108019	0.991489	1.000000	0.509032	0.785987	-0.134502	0.174598	-0.125359	0.322
-0.030773	0.508977	0.509032	1.000000	0.460097	0.064764	0.071041	-0.057127	0.191
-0.087906	0.828273	0.785987	0.460097	1.000000	-0.010322	0.161943	-0.118599	0.294
-0.003839	-0.114238	-0.134502	0.064764	-0.010322	1.000000	0.117074	-0.031895	-0.134
-0.033244	0.198838	0.174598	0.071041	0.161943	0.117074	1.000000	-0.313910	0.132
0.021421	-0.133523	-0.125359	-0.057127	-0.118599	-0.031895	-0.313910	1.000000	0.109
0.004495	0.328665	0.322623	0.191890	0.294350	-0.134961	0.132946	0.109957	1.000
-0.021542	0.018791	0.017978	0.016565	-0.004190	-0.012525	0.010842	-0.012440	-0.004
-0.009808	0.000841	-0.001437	0.026867	0.016115	-0.004750	-0.014436	0.006229	0.025
0.972483	-0.121080	-0.121668	-0.086671	-0.059100	0.002481	-0.037315	0.022424	0.003
0.049776	-0.026319	-0.034245	-0.058972	-0.050016	0.026444	-0.042531	-0.012533	-0.018

In [53]:

```
!pip install matplotlib
```

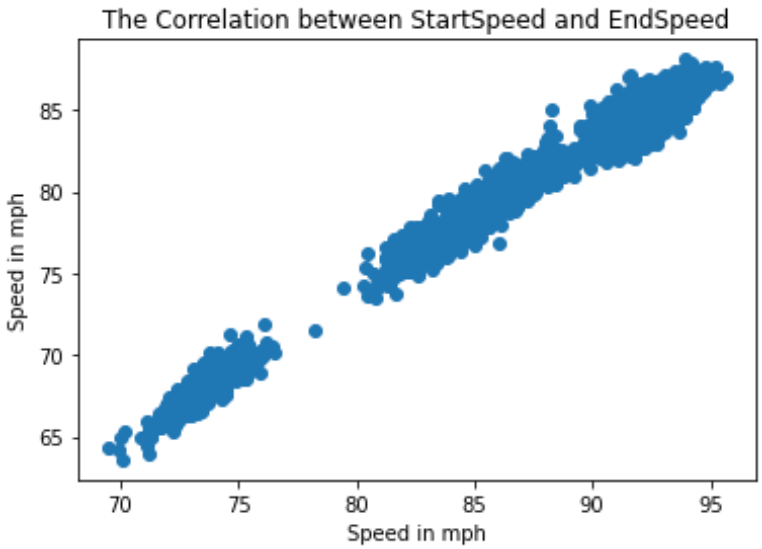
```
Requirement already satisfied: matplotlib in /Users/mayahany/opt/anac
onda3/lib/python3.9/site-packages (3.4.3)
Requirement already satisfied: pillow>=6.2.0 in /Users/mayahany/opt/a
naconda3/lib/python3.9/site-packages (from matplotlib) (8.4.0)
Requirement already satisfied: numpy>=1.16 in /Users/mayahany/opt/ana
conda3/lib/python3.9/site-packages (from matplotlib) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7 in /Users/mayahan
y/opt/anaconda3/lib/python3.9/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/mayahany/op
t/anaconda3/lib/python3.9/site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/mayahany/o
pt/anaconda3/lib/python3.9/site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /Users/mayahany/opt/an
aconda3/lib/python3.9/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: six in /Users/mayahany/opt/anaconda3/l
ib/python3.9/site-packages (from cycler>=0.10->matplotlib) (1.16.0)
```

In [3]:

```
import matplotlib.pyplot as plt
```

In [15]:

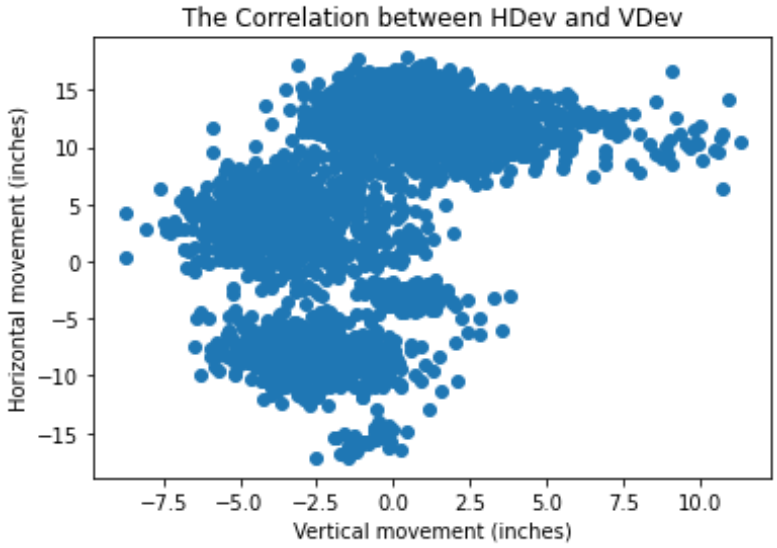
```
plt.scatter(df['StartSpeed'],df['EndSpeed'])
plt.ylabel("Speed in mph")
plt.xlabel("Speed in mph")
plt.title('The Correlation between StartSpeed and EndSpeed')
plt.show()
```



- This graph shows that there is a very strong correlation between StartSpeed and EndSpeed, and the correlation coefficient is 0.991489.

In [16]:

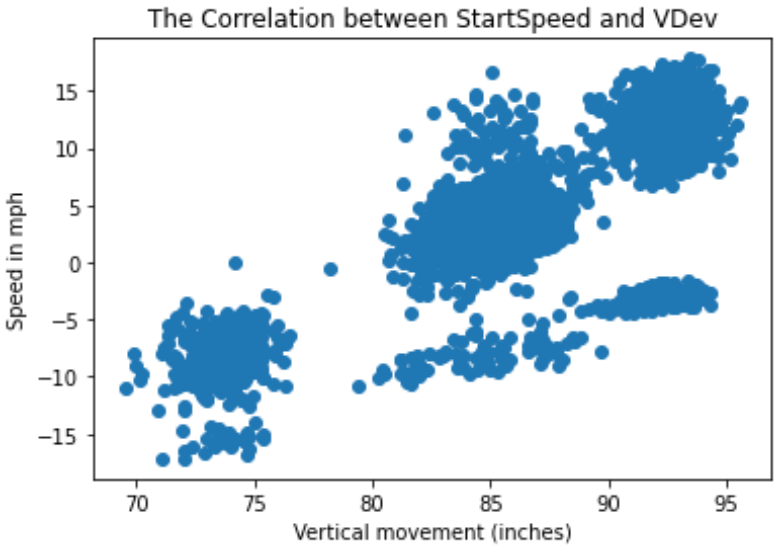
```
plt.scatter(df['HDev'],df['VDev'])
plt.ylabel("Horizontal movement (inches)")
plt.xlabel("Vertical movement (inches)")
plt.title('The Correlation between HDev and VDev')
plt.show()
```



- From the graph, it can be interpreted that there is a moderate correlation between HDev and VDev, and the correlation coefficient is 0.460097.

In [17]:

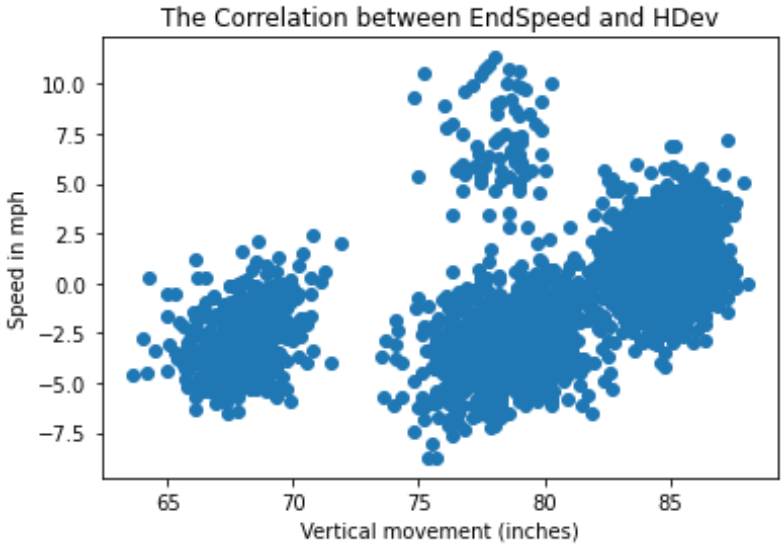
```
plt.scatter(df['StartSpeed'],df['VDev'])
plt.ylabel("Speed in mph")
plt.xlabel("Vertical movement (inches)")
plt.title('The Correlation between StartSpeed and VDev')
plt.show()
```



- In this graph, the large number of data points makes it difficult to fully evaluate the correlation, but the trend is reasonably linear. The correlation coefficient is 0.828273 which is considered a strong correlation.

In [18]:

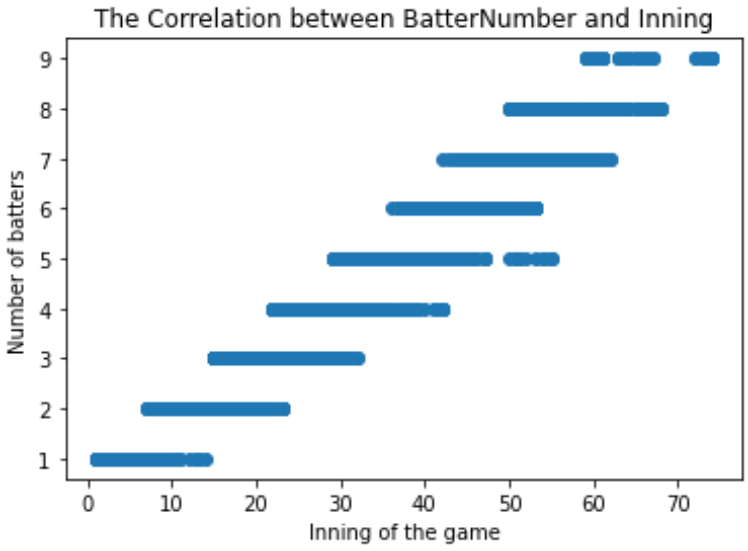
```
plt.scatter(df['EndSpeed'],df['HDev'])
plt.ylabel("Speed in mph")
plt.xlabel("Vertical movement (inches)")
plt.title('The Correlation between EndSpeed and HDev')
plt.show()
```



- This graph shows that there is a weak correlation between EndSpeed and HDev, and the correlation coefficient is 0.509032.

In [19]:

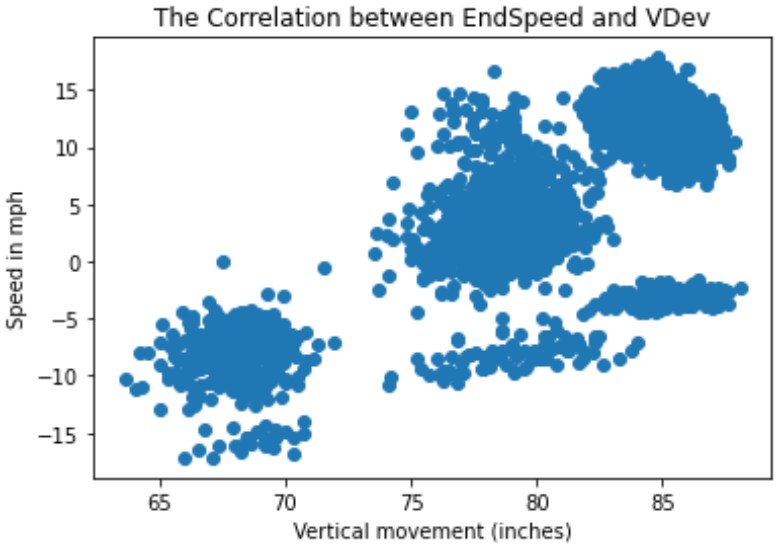
```
plt.scatter(df['BatterNumber'],df['Inning'])
plt.ylabel("Number of batters")
plt.xlabel("Inning of the game")
plt.title('The Correlation between BatterNumber and Inning')
plt.show()
```



- This graph shows that there is a strong correlation between BatterNumber and Inning, and the correlation coefficient is 0.972483.

In [20]:

```
plt.scatter(df['EndSpeed'],df['VDev'])
plt.ylabel("Speed in mph")
plt.xlabel("Vertical movement (inches)")
plt.title('The Correlation between EndSpeed and VDev')
plt.show()
```



- This graph shows that the values are close and be linearly represented, so there is a high moderate (slightly below strong) correlation between EndSpeed and VDev, and the correlation coefficient is 0.785987.

Part 3: Frequency and Percentages

In [46]:

```
freq2=df['Outcome'].value_counts()
freq22=pd.DataFrame({'Outcome':freq2.keys(),'Frequency':freq2.values})
freq22['Percent']=freq22['Frequency']/freq22['Frequency'].sum()*100
freq22
```

Out[46]:

	Outcome	Frequency	Percent
0	Ball	1058	31.099353
1	Called Strike	605	17.783657
2	Foul	605	17.783657
3	In play, out(s)	434	12.757202
4	Swinging Strike	335	9.847149
5	In play, no out	139	4.085832
6	Ball In Dirt	72	2.116402
7	Swinging Strike (Blocked)	51	1.499118
8	In play, run(s)	44	1.293357
9	Foul Tip	35	1.028807
10	Foul Bunt	13	0.382128
11	Foul (Runner Going)	5	0.146972
12	Missed Bunt	3	0.088183
13	Hit By Pitch	3	0.088183

- Here it shows a Frequency table for the value “Outcome” and the percentages next to it. The table has 14 possible results for a pitch, which are ball, called strike, foul, In-play (outs), swinging strike, In-play (no out), ball in the dirt, swinging strike (blocked), in-play(runs), foul tip, foul bunt, foul (runner going), missed bunt and hit by pitch.

In [34]:

```
freq3=df['Class'].value_counts()
freq33=pd.DataFrame({'Class(B=ball, S=strike, or X=in play)':freq3.keys(),'Frequenc
freq33['Percent']=freq33['Frequency']/freq33['Frequency'].sum()*100
freq33
```

Out[34]:

	Class(B=ball, S=strike, or X=in play)	Frequency	Percent
0	S	1652	48.559671
1	B	1133	33.303939
2	X	617	18.136390

- Here it shows a Frequency table for the value “Swing” and the percentages of the values out of 100 on the right.The table has one of three classifications (B=ball, S=strike, or X=in play)

In [35]:

```
freq4=df['Result'].value_counts()
freq44=pd.DataFrame({'Class (Neg=ball or hit, Pos=strike or out)':freq4.keys(),'Fre
freq44['Percent']=freq44['Frequency']/freq44['Frequency'].sum()*100
freq44
```

Out[35]:

	Class (Neg=ball or hit, Pos=strike or out)	Frequency	Percent
0	Pos	2086	61.316872
1	Neg	1316	38.683128

- Here it shows a Frequency table for the value “Result” and the percentages of the values out of 100 on the right. The table is from the pitcher's perspective where Neg is when ball or hit, and Pos is when it is strike or out.

In [48]:

```
freq5=df['Swing'].value_counts()
freq55=pd.DataFrame({'Swing (No or Yes)':freq5.keys(),'Frequency':freq5.values})
freq55['Percent']=freq55['Frequency']/freq55['Frequency'].sum()*100
freq55
```

Out[48]:

	Swing (No or Yes)	Frequency	Percent
0	No	1735	50.999412
1	Yes	1667	49.000588

- Here it shows a Frequency table for the value “Swing” and the percentages of the values out of 100 on the right. The table shows whether the batter swings at the pitch? (No or Yes)

In [47]:

```
freq6=df['PitchType'].value_counts()
freq66=pd.DataFrame({'Pitchtype(CH=changeup, CU=curve, FF=fastball, or SL=slider)':
freq66['Percent']=freq66['Frequency']/freq66['Frequency'].sum()*100
freq66
```

Out[47]:

	Pitchtype(CH=changeup, CU=curve, FF=fastball, or SL=slider)	Frequency	Percent
0	FF	2060	60.552616
1	SL	834	24.514991
2	CU	425	12.492651
3	CH	83	2.439741

- Here it shows a Frequency table for the value “Pitchtype” and the percentages of the values out of 100 on the right. The table shows the code for pitch type, where CH stands for changeup, CU stands for curve, FF stands for fastball, and SL stands for slider.

In [38]:

```
freq7=df['BatterHand'].value_counts()
freq77=pd.DataFrame({'Right/Left':freq7.keys(),'Frequency':freq7.values})
freq77['Percent']=freq77['Frequency']/freq77['Frequency'].sum()*100
freq77
```

Out[38]:

	Right/Left	Frequency	Percent
0	R	2694	79.188713
1	L	708	20.811287

- Here it shows a Frequency table for the value “BatterHand” and the percentages of the values out of 100 on the right. The table shows the number of batters faced so far that game.

Part 4: Contingency Tables

In [7]:

```
table=pd.crosstab(index=df['Swing'],columns=df['StartSpeed'],margins=True)
table
```

Out[7]:

StartSpeed	69.5	69.9	70.0	70.1	70.2	70.9	71.1	71.2	71.3	71.5	...	94.7	94.8	94.9	95.(
Swing															
No	1	1	0	1	1	1	2	2	3	1	...	4	1	1	
Yes	0	0	1	0	0	0	0	0	0	0	...	3	2	0	2
All	1	1	1	1	1	1	2	2	3	1	...	7	3	1	3

3 rows × 211 columns

- This shows that when he does swing the speed is much more consantly higher then when he doesn’t this should mean that he should swing more regularly

In [50]:

```
pd.crosstab(index=df['Class'],columns=df['StrikeCount'],margins=True)
```

Out[50]:

StrikeCount	0	1	2	All
Class				
B	306	492	335	1133
S	430	703	519	1652
X	163	266	188	617
All	899	1461	1042	3402

- This shows that there isnt any corellation between the class of the bat ant the strikes he makes

In [52]:

```
pd.crosstab(index=df['PitchType'],columns=df['Zone'],margins=True)
```

Out[52]:

Zone	1	2	3	4	5	6	7	8	9	11	12	13	14	All
PitchType														
CH	3	4	4	2	3	3	3	5	6	1	7	19	23	83
CU	8	17	17	13	16	19	16	18	17	22	56	120	86	425
FF	138	116	70	158	130	98	110	87	48	402	217	350	136	2060
SL	29	28	21	48	49	28	56	41	18	104	74	258	80	834
All	178	165	112	221	198	148	185	151	89	529	354	747	325	3402

In [61]:

```
pd.crosstab(index=df['BatterHand'],columns=df['PitchType'],margins=True)
```

Out[61]:

PitchType	CH	CU	FF	SL	All
BatterHand					
L	2	77	466	163	708
R	81	348	1594	671	2694
All	83	425	2060	834	3402

- This shows that he uses his right hand much more often which means he is much more confident with his right hand

In [67]:

```
pd.crosstab(index=df['BatterHand'],columns=df['Outs'],margins=True)
```

Out[67]:

Outs	0	1	2	3	All
BatterHand					
L	75	209	258	166	708
R	274	872	835	713	2694
All	349	1081	1093	879	3402

- He still has almost the same ratio of outs whether its with his right or left hand.

In [63]:

```
pd.crosstab(index=df['Batter'],columns=df['Inning'])
```

Out[63]:

Inning	1	2	3	4	5	6	7	8	9
Batter									
A.J. Burnett	0	0	3	0	0	3	0	0	0
A.J. Pollock	10	0	5	0	15	0	4	2	0
Adam LaRoche	7	4	0	3	4	3	4	0	10
Adeiny Hechavarria	0	5	0	2	0	0	2	0	0
Alex Gonzalez	0	0	0	0	0	0	0	5	0
...	...	...	...	...	...	...	...	...	...
Yorvit Torrealba	0	4	0	0	2	0	5	0	0
Yovani Gallardo	0	0	1	0	6	0	0	0	0
Yunel Escobar	0	5	0	0	3	0	6	0	0
Yuniesky Betancourt	0	2	2	0	9	0	0	10	0
Zachary Cozart	0	6	0	3	5	0	8	0	0

206 rows × 9 columns

In [68]:

```
pd.crosstab(index=df['Batter'],columns=df['Outs'],margins=True)
```

Out[68]:

Outs	0	1	2	3	All
Batter					
A.J. Burnett	0	0	6	0	6
A.J. Pollock	6	7	7	16	36
Adam LaRoche	4	7	21	3	35
Adeiny Hechavarria	0	5	2	2	9
Alex Gonzalez	0	0	5	0	5
...	...	...	...	...	...
Yovani Gallardo	0	1	0	6	7
Yunel Escobar	3	0	11	0	14
Yuniesky Betancourt	2	10	11	0	23
Zachary Cozart	4	0	8	10	22
All	349	1081	1093	879	3402

207 rows × 5 columns

- This shows which batter is best for him as it displays who batters with the number of outs he got

4.1 Testing Independence

In [1]:

```
import scipy as stats
from scipy.stats import chi2_contingency
```

In [8]:

```
t1=pd.crosstab(index=df['Swing'],columns=df['StartSpeed'],margins=True)
t11=chi2_contingency(t1)
t11
```

Out[8]:

```
(266.1650582852538,
 0.9999999995313423,
 420,
 array([[5.09994121e-01, 5.09994121e-01, 5.09994121e-01, 5.0999412
1e-01,
         5.09994121e-01, 5.09994121e-01, 1.01998824e+00, 1.0199882
4e+00,
         1.52998236e+00, 5.09994121e-01, 1.52998236e+00, 2.0399764
8e+00,
         2.03997648e+00, 1.01998824e+00, 4.07995297e+00, 1.0199882
4e+00,
         3.05996473e+00, 3.05996473e+00, 4.58994709e+00, 2.0399764
8e+00,
         3.05996473e+00, 5.09994121e+00, 5.60993533e+00, 5.6099353
3e+00,
         6.11992945e+00, 3.05996473e+00, 6.11992945e+00, 7.1399177
0e+00,
         6.62992357e+00, 8.66990006e+00, 8.66990006e+00, 9.6898883
```

The p-value is greater than 0.05 meaning there is no association between the values therefore the values are independent. The degree of freedom is high since there are many variables present. I

In [10]:

```
t2=pd.crosstab(index=df['Class'],columns=df['StrikeCount'],margins=True)
t22=chi2_contingency(t2)
t22
```

Out[10]:

```
(1.1233880625326915,
 0.9990965102476511,
 9,
 array([[ 299.40241035,  486.57054674,  347.02704292, 1133.        ],
        [ 436.55144033,  709.45679012,  505.99176955, 1652.        ],
        [ 163.04614932,  264.97266314,  188.98118754,  617.        ],
        [ 899.        , 1461.        , 1042.        , 3402.
]]))
```

The p-value is greater than 0.05 meaning there is no association between the value therefore the values are independent.. The degree of freedom is low since there are a small number of variables present.

In [11]:

```
t3=pd.crosstab(index=df['PitchType'],columns=df['Zone'],margins=True)
t33=chi2_contingency(t3)
t33
```

Out[11]:

```
(303.99452702830433,
 2.63667602958179e-37,
 52,
 array([[4.34273956e+00, 4.02557319e+00, 2.73251029e+00, 5.39182834e+
00,
         4.83068783e+00, 3.61081717e+00, 4.51352146e+00, 3.68400941e+
00,
         2.17136978e+00, 1.29062316e+01, 8.63668430e+00, 1.82248677e+
01,
         7.92915932e+00, 8.30000000e+01],
        [2.22369195e+01, 2.06128748e+01, 1.39917695e+01, 2.76087596e+
01,
         2.47354497e+01, 1.84891240e+01, 2.31114051e+01, 1.88639036e+
01,
         1.11184597e+01, 6.60861258e+01, 4.42239859e+01, 9.33201058e+
01,
         4.06011170e+01, 4.25000000e+02],
        [1.07783657e+02, 9.99118166e+01, 6.78189300e+01, 1.33821282e+
02,
         1.19894180e+02, 8.96178718e+01, 1.12022340e+02, 9.14344503e+
01,
         5.38918283e+01, 3.20323339e+02, 2.14356261e+02, 4.52328042e+
02,
         1.96796002e+02, 2.06000000e+03],
        [4.36366843e+01, 4.04497354e+01, 2.74567901e+01, 5.41781305e+
01,
         4.85396825e+01, 3.62821869e+01, 4.53527337e+01, 3.70176367e+
01,
         2.18183422e+01, 1.29684303e+02, 8.67830688e+01, 1.83126984e+
02,
         7.96737213e+01, 8.34000000e+02],
        [1.78000000e+02, 1.65000000e+02, 1.12000000e+02, 2.21000000e+
02,
         1.98000000e+02, 1.48000000e+02, 1.85000000e+02, 1.51000000e+
02,
         8.90000000e+01, 5.29000000e+02, 3.54000000e+02, 7.47000000e+
02,
         3.25000000e+02, 3.40200000e+03]]))
```

The p-value is greater than 0.05 meaning there is no association between the values therefore the values are independent. The degree of freedom is a bit high since there are many variables present.

In [12]:

```
t4=pd.crosstab(index=df['BatterHand'],columns=df['PitchType'],margins=True)
t44=chi2_contingency(t4)
t44
```

Out[12]:

```
(23.832939096078324,
 0.0024443255508363277,
 8,
 array([[  17.27336861,   88.44797178,  428.71252205,  173.56613757,
          708.        ],
        [  65.72663139,  336.55202822, 1631.28747795,  660.43386243,
         2694.        ],
        [  83.        ,  425.        , 2060.        ,  834.        ,
         3402.        ]]))
```

The p-value is less than 0.05 meaning there is an association between the values therefore the values are dependent. The degree of freedom is low since there are not as many variables present.

In [13]:

```
t5=pd.crosstab(index=df['BatterHand'],columns=df['Outs'],margins=True)
t55=chi2_contingency(t5)
t55
```

Out[13]:

```
(8.683478205018352,
 0.369696768815235,
 8,
 array([[  72.6313933 ,  224.97001764,  227.46737213,  182.93121693,
          708.        ],
        [ 276.3686067 ,  856.02998236,  865.53262787,  696.06878307,
         2694.        ],
        [ 349.        , 1081.        , 1093.        ,  879.        ,
         3402.        ]]))
```

The p-value is greater than 0.05 meaning there is no association between the values therefore the values are independent. The degree of freedom is low since there are not as many variables present.

In [14]:

```
t6=pd.crosstab(index=df['Batter'],columns=df['Inning'])
t66=chi2_contingency(t6)
t66
```

Out[14]:

```
(6320.337671231693,
 0.0,
 1640,
 array([[0.88536155, 0.83597884, 0.95767196, ..., 0.58024691, 0.32275
132,
         0.0952381 ],
        [5.31216931, 5.01587302, 5.74603175, ..., 3.48148148, 1.93650
794,
         0.57142857],
        [5.16460905, 4.87654321, 5.58641975, ..., 3.38477366, 1.88271
605,
         0.55555556],
        ...,
        [2.06584362, 1.95061728, 2.2345679 , ..., 1.35390947, 0.75308
642,
         0.22222222],
        [3.39388595, 3.20458554, 3.67107584, ..., 2.22427984, 1.23721
34 ,
         0.36507937],
        [3.24632569, 3.06525573, 3.51146384, ..., 2.12757202, 1.18342
152,
         0.34920635]]))
```

The p-value is less than 0.05 and is 0 meaning there is an extreme association between the values therefore the values are dependent. The degree of freedom is high since there are many variables present.

In [15]:

```
t7=pd.crosstab(index=df['Batter'],columns=df['Outs'],margins=True)
t77=chi2_contingency(t7)
t77
```

Out[15]:

```
(2735.3904771926927,
 3.6651126941656044e-203,
 824,
 array([[6.15520282e-01, 1.90652557e+00, 1.92768959e+00, 1.55026455e+
00,
         6.00000000e+00],
        [3.69312169e+00, 1.14391534e+01, 1.15661376e+01, 9.30158730e+
00,
         3.60000000e+01],
        [3.59053498e+00, 1.11213992e+01, 1.12448560e+01, 9.04320988e+
00,
         3.50000000e+01],
        ...,
        [2.35949442e+00, 7.30834803e+00, 7.38947678e+00, 5.94268078e+
00,
         2.30000000e+01],
        [2.25690770e+00, 6.99059377e+00, 7.06819518e+00, 5.68430335e+
00,
         2.20000000e+01],
        [3.49000000e+02, 1.08100000e+03, 1.09300000e+03, 8.79000000e+
02,
         3.40200000e+03]]))
```

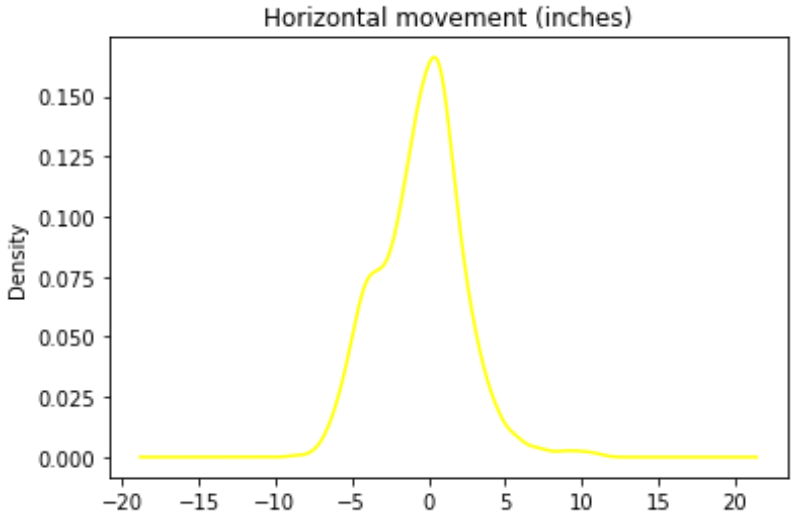
The p-value is less than 0.05 meaning there is an association between the values therefore the values are dependent. The degree of freedom is high since there are many variables present.

Part 5: Graphical Displays

1. Density Graphs

In [4]:

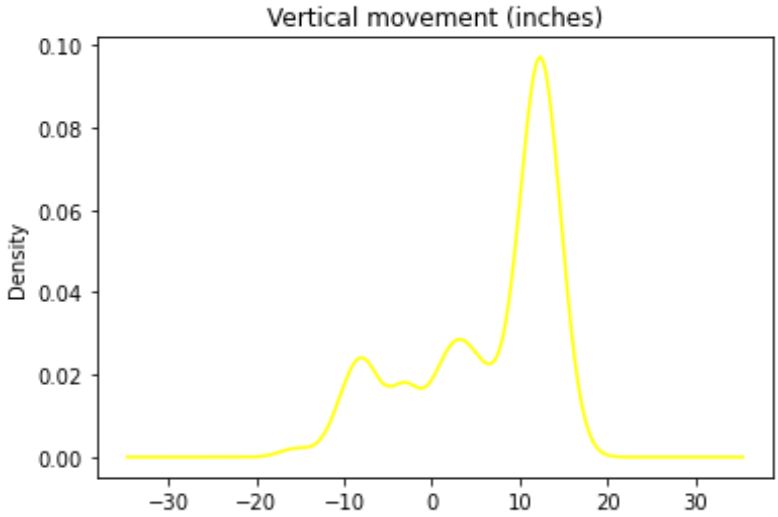
```
df.HDev.plot.density(color='yellow')
plt.title('Horizontal movement (inches)')
plt.show()
```



Here it shows the density plot for the value “HDev”. Hdev is the Horizontal movement in inches. This graph has a semi-bimodal distribution, and it is not skewed so the mean is equal to the median

In [5]:

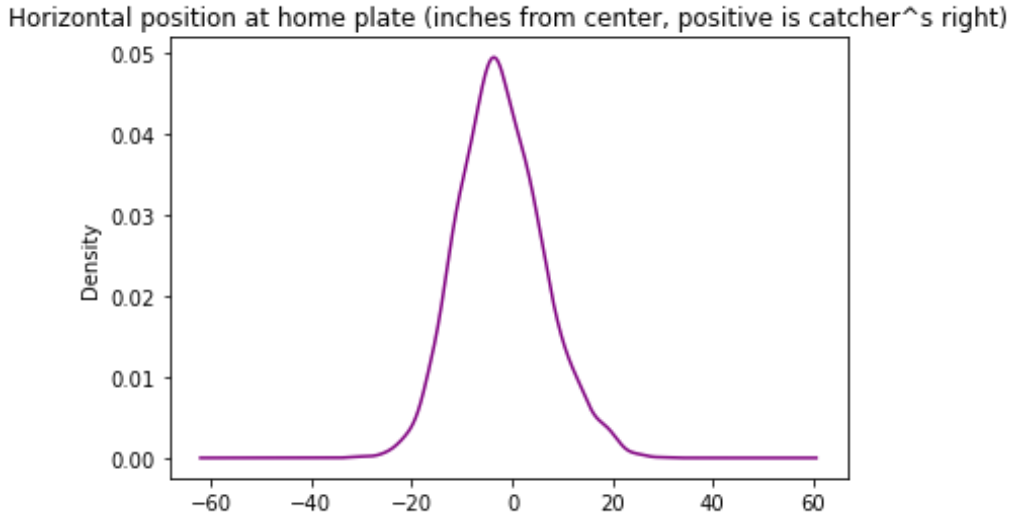
```
df.VDev.plot.density(color='yellow')
plt.title('Vertical movement (inches)')
plt.show()
```



Here it shows the density plot for the value “VDev”. VDev is Vertical movement in inches. This graph has a multimodal distribution, and it is slightly left-skewed so the mean is less than the median.

In [12]:

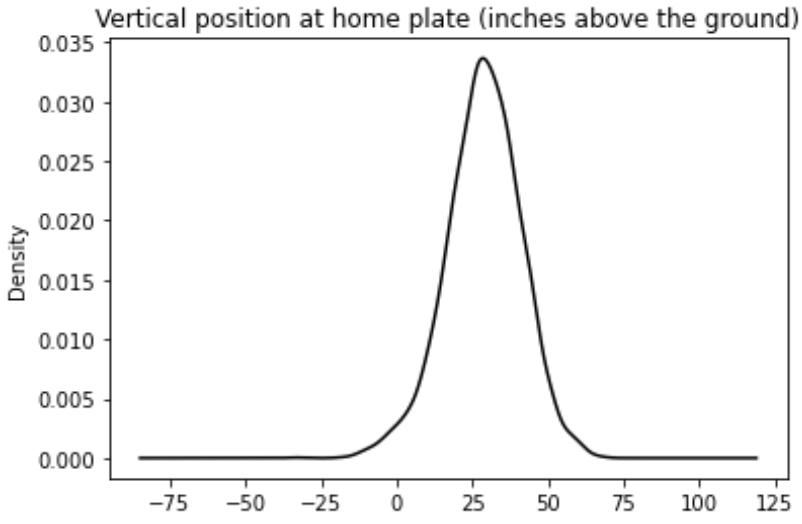
```
df.HPos.plot.density(color='purple')
plt.title('Horizontal position at home plate (inches from center, positive is catch
plt.show()
```



Here it shows the density plot for the value “HPos”. HPos is the horizontal position at home plate (inches from the center, positive is catcher's right) This graph has a unimodal distribution, and it is slightly right-skewed so the mean is greater than the median.

In [13]:

```
df.VPos.plot.density(color='black')
plt.title('Vertical position at home plate (inches above the ground)')
plt.show()
```

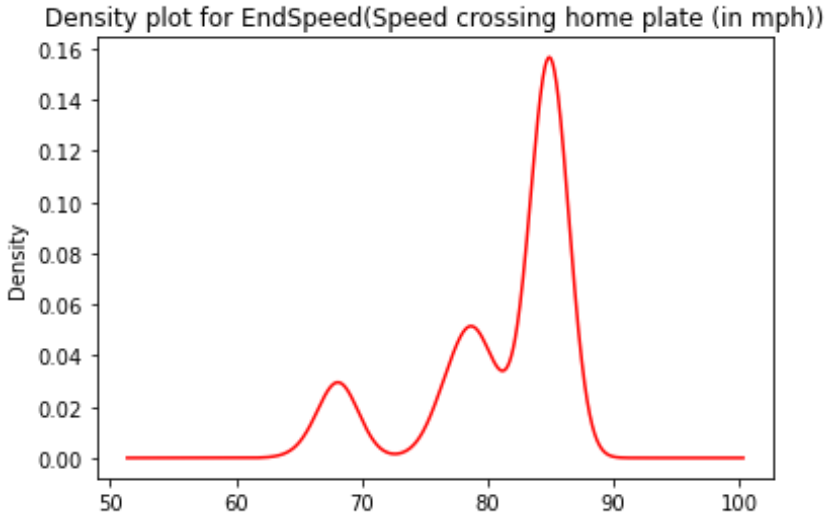


Here it shows the density plot for the value “VPos”. HPos is vertical position at home plate (inches above the

ground) This graph has a unimodal distribution, and it is slightly left-skewed so the mean is less than the median.

In [12]:

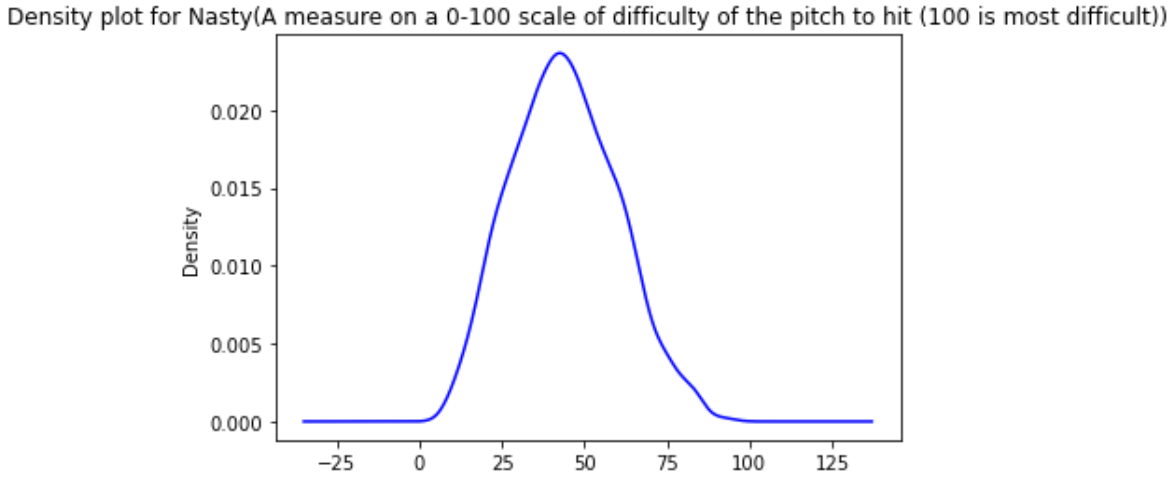
```
df.EndSpeed.plot.density(color='red')
plt.title('Density plot for EndSpeed(Speed crossing home plate (in mph))')
plt.show()
```



- Here it shows the density plot for the value “EndSpeed”. Endspeed is speed crossing home plate (in mph).This graph has a multimodal distribution, and it is slightly left-skewed so the mean is less than the median.

In [11]:

```
df.Nasty.plot.density(color='blue')
plt.title('Density plot for Nasty(A measure on a 0-100 scale of difficulty of the p
plt.show()
```

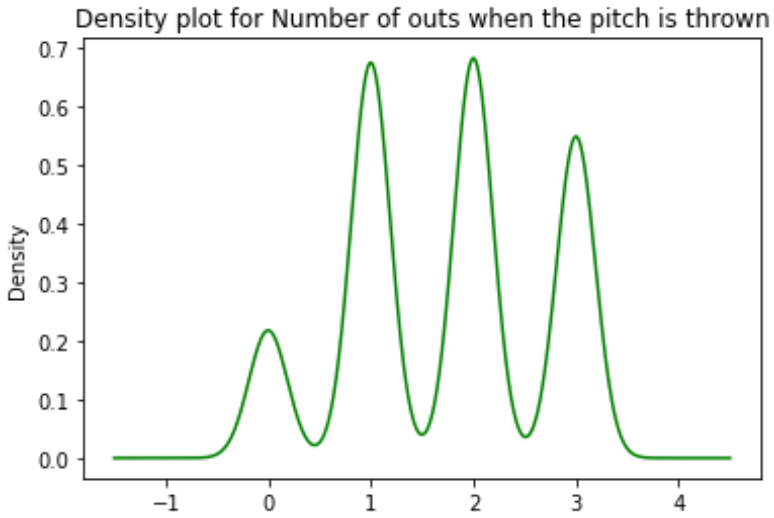


- Here it shows the density plot for the value “Nasty”. Nasty is a measure on a 0-100 scale of difficulty of the pitch to hit (100 is most difficult). This graph has a unimodal distribution, and it is left skewed

meaning the mean is less than the median.

In [12]:

```
df.Outs.plot.density(color='green')
plt.title('Density plot for Number of outs when the pitch is thrown')
plt.show()
```



- Here it shows the density plot for the value “Outs”. Outs are the number of outs when the pitch is thrown. This graph has a multimodal distribution and does appear to have a chaotic pattern to it.

In [16]:

```
freq5=df['Swing'].value_counts()
freq8=df['InningSide'].value_counts()
pd.crosstab(df.Swing,df.InningSide)
```

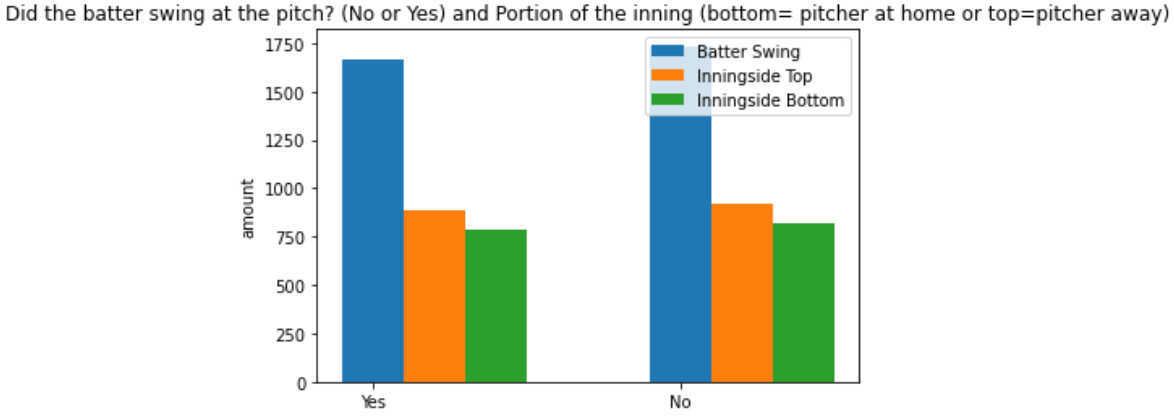
Out[16]:

InningSide	bottom	top
Swing		
No	816	919
Yes	782	885

2. Bar plots

In [14]:

```
import numpy as np
swing=['Yes','No']
ypos= np.arange(len(swing))
plt.xticks(ypos,swing)
plt.ylabel('amount')
plt.title('Did the batter swing at the pitch? (No or Yes) and Portion of the inning
number=[1667,1735]
InningsideB=[782,816]
InningsideT=[885,919]
plt.bar(ypos,number,width=0.2,label='Batter Swing')
plt.bar(ypos+0.2,InningsideT,width=0.2,label='Inningside Top')
plt.bar(ypos+0.4,InningsideB,width=0.2,label='Inningside Bottom')
plt.legend()
plt.show()
```



- For this bar plot. It has three values, the swing, the top inning side, and the bottom inning side. The batter swing is blue, the inning side top is orange and the inning side is green. In this plot, amount of times the batter swing at the pitch is less than it did not. And the inning side and top and bottom are more when the batter didn't swing at the pitch.

In [40]:

```
freq88=df['BatterHand'].value_counts()
pd.crosstab(df.ABEvent,df.BatterHand)
```

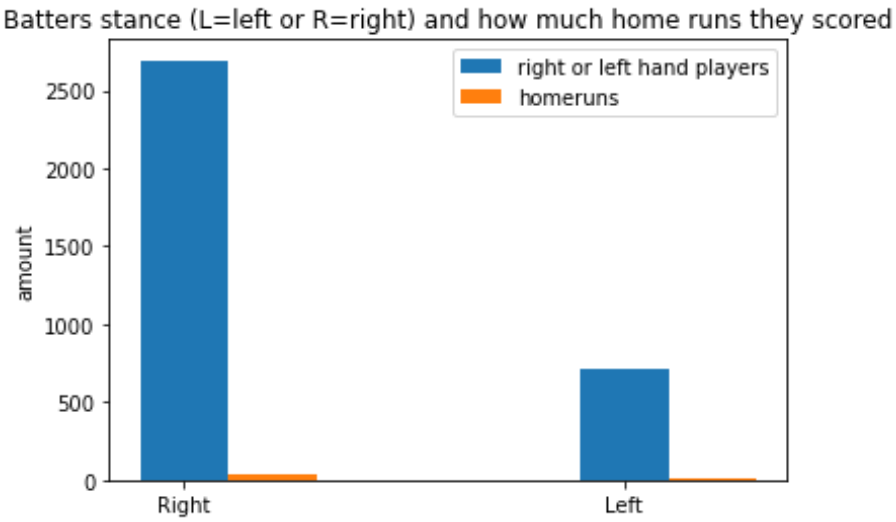
Out[40]:

BatterHand	L	R
ABEvent		
Bunt Groundout	3	2
Bunt Lineout	0	1
Catcher Interference	0	10
Double	26	73
Double Play	0	22
Field Error	10	22
Fielders Choice	0	3
Fielders Choice Out	0	2
Flyout	48	254
Forceout	2	34
Grounded Into DP	8	47
Groundout	92	552
Hit By Pitch	5	4
Home Run	4	39
Intent Walk	0	5
Lineout	39	148
Pop Out	15	142
Runner Out	3	3
Sac Bunt	1	9
Sac Fly	0	6
Single	79	337
Strikeout	301	743
Triple	5	6
Walk	67	230

- Here is a contingency table that displays whether each player is right-handed or left-handed, and how it affects the result of the at-bat which is a value called “ABEevent”. From these values, we will take the home run and whether the player is right-handed or not.

In [43]:

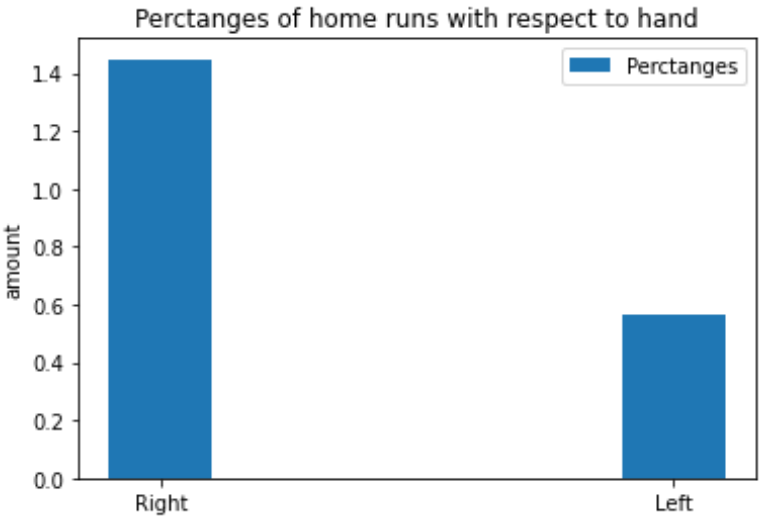
```
batterhand=['Right','Left']
ypos= np.arange(len(batterhand))
plt.xticks(ypos,batterhand)
plt.ylabel('amount')
plt.title('Batters stance (L=left or R=right) and how much home runs they scored')
RL=[2694,708]
HR=[39,4]
plt.bar(ypos,RL,width=0.2,label='right or left hand players')
plt.bar(ypos+0.2,HR,width=0.2,label='homeruns')
plt.legend()
plt.show()
```



- Here because the graph is so unbalanced since there is more right-hand player than left hand. So it fix it, we change it into percentages.

In [45]:

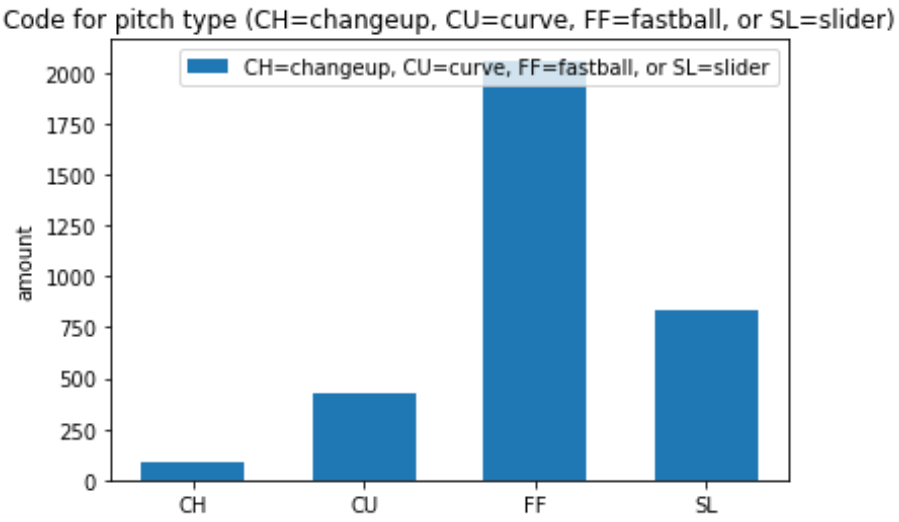
```
batterhand=['Right','Left']
ypos= np.arange(len(batterhand))
plt.xticks(ypos,batterhand)
plt.ylabel('amount')
plt.title('Perctanges of home runs with respect to hand')
PR=[(39/2694)*100,(4/708)*100]
plt.bar(ypos,PR,width=0.2,label='Perctanges')
plt.legend()
plt.show()
```



- Now we could conclude that the right-handed players score more home runs than the left.

In [49]:

```
freq6=df['PitchType'].value_counts()
pitchtype=['CH','CU','FF','SL']
ypos= np.arange(len(pitchtype))
plt.xticks(ypos,pitchtype)
plt.ylabel('amount')
plt.title('Code for pitch type (CH=changeup, CU=curve, FF=fastball, or SL=slider)')
N=[83,425,2060,834]
plt.bar(ypos,N,width=0.6,label='CH=changeup, CU=curve, FF=fastball, or SL=slider')
plt.legend()
plt.show()
```



- Here it shows the amount of each pitch type, either, changeup, curve, fastball, or slider. Fastball had the highest amount.

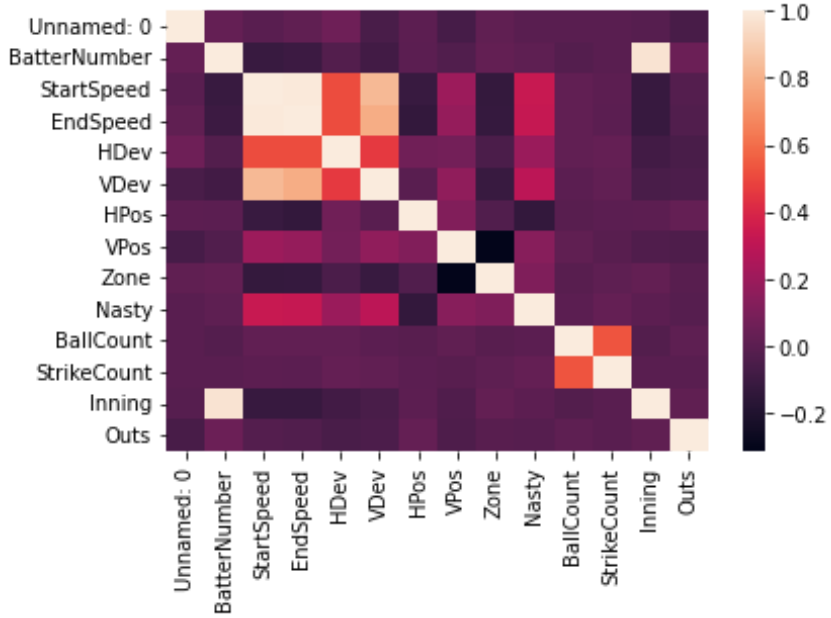
3. Heatmap

In [37]:

```
df=pd.read_csv('Kershaw.csv')
```

In [43]:

```
import seaborn as sns
x=df.corr()
sns.heatmap(x)
plt.show()
```



- As shown above the Heatmap is displayed and it is represented as two-dimensional grid representation. It shows the correlation which is calculated using the corr() function. The corr() function returns the correlation matrix. This correlation matrix is plotted using the heatmap() function for the grid view of the correlation matrix. It takes two parameters: the correlation matrix and annot. The annot parameter is passed as True. So, this graph shows the correlations among all variables from scale -0.01 till 1, and 1 represent perfect correlation while 0 represents that there is no correlation and negative values represents that there is negative correlation.

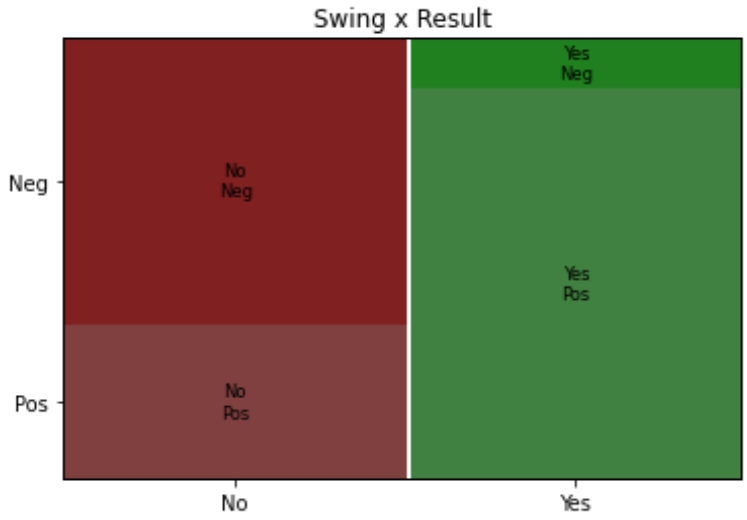
4. Mosaic Plots

In [1]:

```
import matplotlib.pyplot as plt
from statsmodels.graphics.mosaicplot import mosaic
```

In [5]:

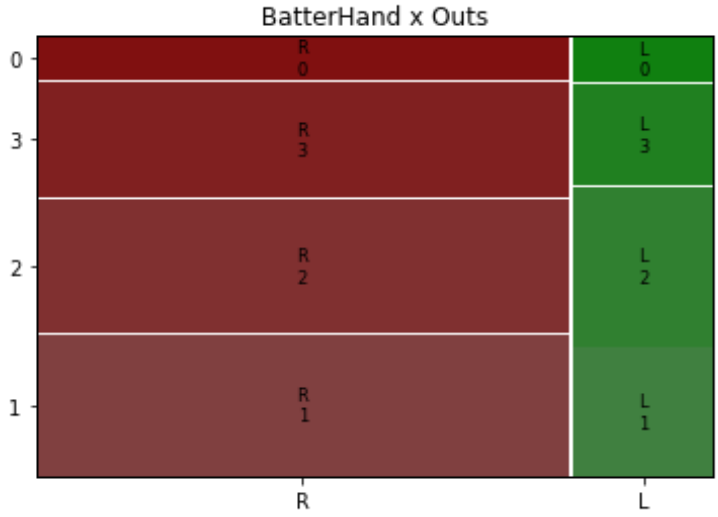
```
mosaic(df, ['Swing', 'Result'], title=' Swing x Result ')
plt.show()
```



- This shows if he swings or not what will the out come be so here it shows that when he swings mostly he gets a positive and when he doesnt he gets a negative result so he should swing more often.

In [12]:

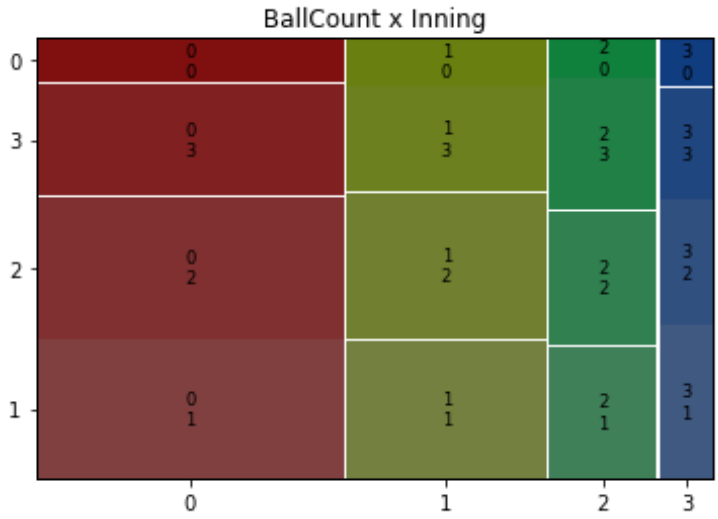
```
mosaic(df, ['BatterHand', 'Outs'], title=' BatterHand x Outs ')
plt.show()
```



- This shows when he uses his left hand and when he uses his right the diffrence in the number of outs he commits.

In [8]:

```
mosaic(df, ['BallCount', 'Outs'], title=' BallCount x Inning ')
plt.show()
```



- This plot shows how many ball counts he used comapred with how much went in.

--End of the assignment-- THANK YOU!! 🤗👩‍💻👨‍💻🌟