

# batting\_pre

February 27, 2020

```
[969]: import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# We're going to be reassigning some columns, so we'll turn off this warning -L
→we know what we're doing!
pd.options.mode.chained_assignment = None # default='warn'
```

```
[970]: df = pd.read_csv('./data/lahman/mlb_data/Batting.csv').sort_values('playerID')
```

```
[971]: df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

Cleaning the Data - Missing Values

Print percentages of missing data in each column of the batting table

```
[972]: 100 * df.isnull().sum() / len(df)
```

```
[972]: retroID      0.000000
yearID      0.000000
stint       0.000000
teamID      0.000000
lgID        0.000000
G           0.000000
AB          0.000000
R           0.000000
H           0.000000
2B          0.000000
3B          0.000000
HR          0.000000
RBI         0.000000
SB          0.000000
CS          8.221708
BB          0.000000
SO          0.000000
IBB        21.711883
HBP         0.000000
```

```
SH          0.000000
SF          21.090864
GIDP        9.839985
dtype: float64
```

Since this data is by season, it's likely that we have entries for a player for one season with no data in these fields but there is data for other seasons. Since we're taking aggregate sums for each player, we have two options: set these null values to zero so they don't add to the sum, or set them to the average for that player. We'll have to test the theory to see which is more viable.

We're going to start with IBB rather than CS, since it's a more significant chunk of the dataset.

Handling missing IBB data

```
[973]: df[(df['IBB'].isnull())]
```

```
[973]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
19269	aaronha01	1954	1	ML1	NL	122	468	58	131	27	...	69	
18684	aberal01	1953	2	DET	AL	17	23	2	3	0	...	2	
16858	aberal01	1950	1	CLE	AL	1	2	0	0	0	...	0	
19270	aberal01	1954	1	DET	AL	32	39	3	5	0	...	3	
18683	aberal01	1953	1	CLE	AL	6	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
15128	zuberbi01	1946	1	NYA	AL	3	2	0	0	0	...	0	
14447	zuberbi01	1945	1	NYA	AL	21	42	1	7	0	...	3	
13868	zuberbi01	1944	1	NYA	AL	22	31	1	4	0	...	1	
19843	zuverge01	1954	1	CIN	NL	2	2	1	1	0	...	0	
19844	zuverge01	1954	2	DET	AL	35	64	1	8	1	...	3	

  

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
19269	2	2.0	28	39	NaN	3	6	4.0	13.0
18684	0	0.0	1	6	NaN	0	1	NaN	0.0
16858	0	0.0	1	1	NaN	0	0	NaN	0.0
19270	0	0.0	2	17	NaN	0	3	1.0	1.0
18683	0	0.0	2	0	NaN	0	0	NaN	0.0
...	...	...	...	...	...	...	...	...	...
15128	0	0.0	0	1	NaN	0	0	NaN	0.0
14447	0	0.0	1	13	NaN	0	2	NaN	1.0
13868	0	0.0	0	10	NaN	0	4	NaN	1.0
19843	0	0.0	0	1	NaN	0	0	0.0	0.0
19844	0	1.0	1	14	NaN	0	9	0.0	2.0

[19159 rows x 22 columns]

```
[974]: df[(df['retroID'] == 'aberal01')]
```

```
[974]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	\
19846	aberal01	1955	1	DET	AL	39	17	0	1	0	...	0	0	

18684	aberal01	1953	2	DET	AL	17	23	2	3	0	...	2	0
16858	aberal01	1950	1	CLE	AL	1	2	0	0	0	...	0	0
19270	aberal01	1954	1	DET	AL	32	39	3	5	0	...	3	0
18683	aberal01	1953	1	CLE	AL	6	0	0	0	0	...	0	0
21123	aberal01	1957	2	KC1	AL	3	1	0	1	0	...	0	0
20501	aberal01	1956	1	DET	AL	42	10	0	3	0	...	0	0
21122	aberal01	1957	1	DET	AL	28	8	0	1	0	...	1	0

	CS	BB	SO	IBB	HBP	SH	SF	GIDP
19846	0.0	0	9	0.0	0	2	0.0	1.0
18684	0.0	1	6	NaN	0	1	NaN	0.0
16858	0.0	1	1	NaN	0	0	NaN	0.0
19270	0.0	2	17	NaN	0	3	1.0	1.0
18683	0.0	2	0	NaN	0	0	NaN	0.0
21123	0.0	0	0	0.0	0	0	0.0	0.0
20501	0.0	1	4	0.0	0	2	0.0	0.0
21122	0.0	1	4	0.0	0	0	0.0	0.0

[8 rows x 22 columns]

```
[975]: df[(df['retroID'] == 'zuberbi01')]
```

```
[975]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	\
9445	zuberbi01	1936	1	CLE	AL	2	5	1	1	0	...	0	0	
10501	zuberbi01	1938	1	CLE	AL	15	7	0	0	0	...	0	0	
11080	zuberbi01	1939	1	CLE	AL	16	5	0	1	0	...	0	0	
11621	zuberbi01	1940	1	CLE	AL	17	3	0	1	0	...	0	0	
12203	zuberbi01	1941	1	WS1	AL	36	26	0	0	0	...	0	0	
12742	zuberbi01	1942	1	WS1	AL	37	39	5	6	3	...	3	0	
13299	zuberbi01	1943	1	NYA	AL	20	38	1	7	1	...	2	0	
15711	zuberbi01	1947	1	BOS	AL	20	13	0	2	0	...	0	0	
15129	zuberbi01	1946	2	BOS	AL	15	18	1	2	0	...	2	0	
15128	zuberbi01	1946	1	NYA	AL	3	2	0	0	0	...	0	0	
14447	zuberbi01	1945	1	NYA	AL	21	42	1	7	0	...	3	0	
13868	zuberbi01	1944	1	NYA	AL	22	31	1	4	0	...	1	0	

	CS	BB	SO	IBB	HBP	SH	SF	GIDP
9445	0.0	0	1	NaN	0	0	NaN	NaN
10501	0.0	0	1	NaN	0	1	NaN	NaN
11080	0.0	0	2	NaN	0	0	NaN	0.0
11621	0.0	0	0	NaN	0	0	NaN	0.0
12203	0.0	1	8	NaN	0	2	NaN	0.0
12742	0.0	1	7	NaN	0	3	NaN	2.0
13299	0.0	4	14	NaN	0	5	NaN	2.0
15711	0.0	2	3	NaN	0	2	NaN	2.0
15129	0.0	1	6	NaN	0	1	NaN	0.0
15128	0.0	0	1	NaN	0	0	NaN	0.0

```
14447  0.0   1  13  NaN   0   2  NaN   1.0
13868  0.0   0  10  NaN   0   4  NaN   1.0
```

```
[12 rows x 22 columns]
```

First let's look at IBB, intentional bases on balls. It seems like most of the missing data is from early in the dataset - it could be that IBB was not recorded then, and/or not considered a trackable play?

```
[976]: df[(df['IBB'].isnull())]['yearID'].max()
```

```
[976]: 1954
```

```
[977]: df[(df['IBB'].isnull())]['yearID'].min()
```

```
[977]: 1919
```

```
[978]: df[(df['IBB'].isnull())]
```

```
[978]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
19269	aaronha01	1954	1	ML1	NL	122	468	58	131	27	...	69	
18684	aberal01	1953	2	DET	AL	17	23	2	3	0	...	2	
16858	aberal01	1950	1	CLE	AL	1	2	0	0	0	...	0	
19270	aberal01	1954	1	DET	AL	32	39	3	5	0	...	3	
18683	aberal01	1953	1	CLE	AL	6	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
15128	zuberbi01	1946	1	NYA	AL	3	2	0	0	0	...	0	
14447	zuberbi01	1945	1	NYA	AL	21	42	1	7	0	...	3	
13868	zuberbi01	1944	1	NYA	AL	22	31	1	4	0	...	1	
19843	zuverge01	1954	1	CIN	NL	2	2	1	1	0	...	0	
19844	zuverge01	1954	2	DET	AL	35	64	1	8	1	...	3	

  

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
19269	2	2.0	28	39	NaN	3	6	4.0	13.0
18684	0	0.0	1	6	NaN	0	1	NaN	0.0
16858	0	0.0	1	1	NaN	0	0	NaN	0.0
19270	0	0.0	2	17	NaN	0	3	1.0	1.0
18683	0	0.0	2	0	NaN	0	0	NaN	0.0
...	...	...	...	...	...	...	...	...	...
15128	0	0.0	0	1	NaN	0	0	NaN	0.0
14447	0	0.0	1	13	NaN	0	2	NaN	1.0
13868	0	0.0	0	10	NaN	0	4	NaN	1.0
19843	0	0.0	0	1	NaN	0	0	0.0	0.0
19844	0	1.0	1	14	NaN	0	9	0.0	2.0

```
[19159 rows x 22 columns]
```

We have 19159 total rows where there is no data for IBB, and we know none of those rows goes

past the year 1954...

```
[979]: df[(df['yearID'] < 1955)]
```

```
[979]:      retroID  yearID  stint teamID lgID   G  AB  R   H  2B  ...  RBI  \
19269  aaronha01   1954     1    ML1   NL  122  468  58  131  27  ...   69
18684  aberal01   1953     2    DET   AL   17   23   2    3   0  ...    2
16858  aberal01   1950     1    CLE   AL    1    2   0    0   0  ...    0
19270  aberal01   1954     1    DET   AL   32   39   3    5   0  ...    3
18683  aberal01   1953     1    CLE   AL    6    0   0    0   0  ...    0
...      ...      ...      ...      ...      ...      ...      ...      ...      ...      ...      ...
13868  zuberbi01  1944     1    NYA   AL   22   31   1    4   0  ...    1
18682  zuverge01  1952     1    CLE   AL    2    0   1    0   0  ...    0
19843  zuverge01  1954     1    CIN   NL    2    2   1    1   0  ...    0
19844  zuverge01  1954     2    DET   AL   35   64   1    8   1  ...    3
18050  zuverge01  1951     1    CLE   AL   16    0   0    0   0  ...    0
```

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
19269	2	2.0	28	39	NaN	3	6	4.0	13.0
18684	0	0.0	1	6	NaN	0	1	NaN	0.0
16858	0	0.0	1	1	NaN	0	0	NaN	0.0
19270	0	0.0	2	17	NaN	0	3	1.0	1.0
18683	0	0.0	2	0	NaN	0	0	NaN	0.0
...	..	...	..	..	...	...	..	...	...
13868	0	0.0	0	10	NaN	0	4	NaN	1.0
18682	0	0.0	0	0	0.0	0	0	0.0	0.0
19843	0	0.0	0	1	NaN	0	0	0.0	0.0
19844	0	1.0	1	14	NaN	0	9	0.0	2.0
18050	0	0.0	0	0	0.0	0	0	0.0	0.0

[19845 rows x 22 columns]

And we have 19845 total rows up to the year 1954. That means...

```
[980]: 19159 / 19845
```

```
[980]: 0.9654320987654321
```

Over 96% of the data before 1955 is missing IBB. I think this gives justification to just setting all of those NaNs to 0.

```
[981]: df['IBB'].fillna(value=0, inplace=True)
```

```
[982]: 100 * df.isnull().sum() / len(df)
```

```
[982]: retroID    0.000000
      yearID    0.000000
      stint    0.000000
```

```

teamID      0.000000
lgID        0.000000
G            0.000000
AB          0.000000
R            0.000000
H            0.000000
2B          0.000000
3B          0.000000
HR          0.000000
RBI         0.000000
SB          0.000000
CS          8.221708
BB          0.000000
SO          0.000000
IBB         0.000000
HBP         0.000000
SH          0.000000
SF          21.090864
GIDP        9.839985
dtype: float64

```

Our IBB issue is solved. Let's move on to SF (sacrifice flies). We'll check the years and rows again to see if we're justified in using the same method to eliminate nulls.

Handling missing SF data

```
[983]: df[(df['SF'].isnull())]['yearID'].max()
```

```
[983]: 1953
```

```
[984]: df[(df['SF'].isnull())]['yearID'].min()
```

```
[984]: 1919
```

```
[985]: df[(df['SF'].isnull())].shape[0]
```

```
[985]: 18611
```

```
[986]: df[(df['yearID'] < 1954)].shape[0]
```

```
[986]: 19269
```

```
[987]: 18611/19269
```

```
[987]: 0.965851886449738
```

Almost the same percentage, and one less year covered. I think we can fill those missing values with 0.

```
[988]: df['SF'].fillna(value=0, inplace=True)
```

```
[989]: 100 * df.isnull().sum() / len(df)
```

```
[989]: retroID      0.000000
      yearID      0.000000
      stint      0.000000
      teamID      0.000000
      lgID        0.000000
      G           0.000000
      AB          0.000000
      R           0.000000
      H           0.000000
      2B          0.000000
      3B          0.000000
      HR          0.000000
      RBI         0.000000
      SB          0.000000
      CS          8.221708
      BB          0.000000
      SO          0.000000
      IBB         0.000000
      HBP         0.000000
      SH          0.000000
      SF          0.000000
      GDP         9.839985
      dtype: float64
```

Two more to go, let's move on to CS (caught stealing)

Handling missing CS data

```
[990]: df[(df['CS'].isnull())]['yearID'].max()
```

```
[990]: 1950
```

```
[991]: df[(df['CS'].isnull())]['yearID'].min()
```

```
[991]: 1919
```

```
[992]: df[(df['CS'].isnull())]
```

```
[992]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
14448	abernwo01	1946	1	NY1	NL	15	8	0	0	0	...	0	
16285	aberscl01	1949	1	CHN	NL	4	7	0	0	0	...	0	
15712	aberscl01	1948	1	CHN	NL	12	32	1	6	1	...	6	
15131	aberscl01	1947	1	CHN	NL	47	140	24	39	6	...	20	
16286	abramca01	1949	1	BRO	NL	8	24	6	2	1	...	0	

...	...	...	...	...	...	...	...	...	...	...	...	...
532	zitzmbi01	1919	1	PIT	NL	11	26	5	5	1	...	2
4782	zitzmbi01	1927	1	CIN	NL	88	232	47	66	10	...	24
5312	zitzmbi01	1928	1	CIN	NL	101	266	53	79	9	...	33
533	zitzmbi01	1919	2	CIN	NL	2	1	0	0	0	...	0
5842	zitzmbi01	1929	1	CIN	NL	47	84	18	19	3	...	6

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
14448	0	NaN	0	4	0.0	0	0	0.0	0.0
16285	0	NaN	0	2	0.0	0	0	0.0	1.0
15712	0	NaN	5	10	0.0	0	0	0.0	0.0
15131	0	NaN	20	32	0.0	0	0	0.0	5.0
16286	1	NaN	7	6	0.0	0	0	0.0	1.0
...	...	...	...	...	...	...	...	...	...
532	2	NaN	0	6	0.0	0	1	0.0	NaN
4782	9	NaN	20	18	0.0	4	17	0.0	NaN
5312	13	NaN	13	22	0.0	3	14	0.0	NaN
533	0	NaN	0	0	0.0	0	0	0.0	NaN
5842	4	NaN	9	10	0.0	1	2	0.0	NaN

[7255 rows x 22 columns]

```
[993]: df[(df['retroID'] == 'zitzmbi01')]
```

```
[993]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
4241	zitzmbi01	1926	1	CIN	NL	53	94	21	23	2	...	3	
532	zitzmbi01	1919	1	PIT	NL	11	26	5	5	1	...	2	
3715	zitzmbi01	1925	1	CIN	NL	104	301	53	76	13	...	21	
4782	zitzmbi01	1927	1	CIN	NL	88	232	47	66	10	...	24	
5312	zitzmbi01	1928	1	CIN	NL	101	266	53	79	9	...	33	
533	zitzmbi01	1919	2	CIN	NL	2	1	0	0	0	...	0	
5842	zitzmbi01	1929	1	CIN	NL	47	84	18	19	3	...	6	

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
4241	3	NaN	6	7	0.0	2	3	0.0	NaN
532	2	NaN	0	6	0.0	0	1	0.0	NaN
3715	11	11.0	35	22	0.0	6	2	0.0	NaN
4782	9	NaN	20	18	0.0	4	17	0.0	NaN
5312	13	NaN	13	22	0.0	3	14	0.0	NaN
533	0	NaN	0	0	0.0	0	0	0.0	NaN
5842	4	NaN	9	10	0.0	1	2	0.0	NaN

[7 rows x 22 columns]

```
[994]: df[(df['CS'].isnull())].shape[0]
```

```
[994]: 7255
```



```
[995]: df[(df['yearID'] < 1951)].shape[0]
```

```
[995]: 17435
```

```
[996]: 7255/17435
```

```
[996]: 0.4161170060223688
```

There isn't a great solution for this. If we drop all missing rows with NaN for CS, we're going to lose over 41% of the data prior to 1951. It doesn't encompass enough of the data to just fill in values like we did before, we can't drop rows, and we don't want to drop the column since it isn't missing any data after 1950. One idea, and this may be controversial, is to find the average ratio between SB (stolen bases) and CS and fill in with values based on that ratio.

First, we'll get all data without missing CS values

```
[997]: df_temp = df[(df['CS'].notnull())]
# df_temp
```

```
[998]: total_sb = df_temp['SB'].sum()
total_sb
```

```
[998]: 182622
```

```
[999]: total_cs = df_temp['CS'].sum()
total_cs
```

```
[999]: 94186.0
```

```
[1000]: total_sb/total_cs
```

```
[1000]: 1.9389505871360924
```

So on average, players are almost twice as likely to steal a base as they are to get caught. This is easy math that we're going to round to make it even easier. It's probably not the best method of solving this issue but at least we still have over 60 years of clean data!

```
[1001]: df[(df['CS'].isnull())]
```

```
[1001]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
14448	abernwo01	1946	1	NY1	NL	15	8	0	0	0	...	0	
16285	aberscl01	1949	1	CHN	NL	4	7	0	0	0	...	0	
15712	aberscl01	1948	1	CHN	NL	12	32	1	6	1	...	6	
15131	aberscl01	1947	1	CHN	NL	47	140	24	39	6	...	20	
16286	abramca01	1949	1	BRO	NL	8	24	6	2	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
532	zitzmbi01	1919	1	PIT	NL	11	26	5	5	1	...	2	
4782	zitzmbi01	1927	1	CIN	NL	88	232	47	66	10	...	24	

5312	zitzmbi01	1928	1	CIN	NL	101	266	53	79	9	...	33
533	zitzmbi01	1919	2	CIN	NL	2	1	0	0	0	...	0
5842	zitzmbi01	1929	1	CIN	NL	47	84	18	19	3	...	6

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
14448	0	NaN	0	4	0.0	0	0	0.0	0.0
16285	0	NaN	0	2	0.0	0	0	0.0	1.0
15712	0	NaN	5	10	0.0	0	0	0.0	0.0
15131	0	NaN	20	32	0.0	0	0	0.0	5.0
16286	1	NaN	7	6	0.0	0	0	0.0	1.0
...	..	..	..	..	...	...	..	...	...
532	2	NaN	0	6	0.0	0	1	0.0	NaN
4782	9	NaN	20	18	0.0	4	17	0.0	NaN
5312	13	NaN	13	22	0.0	3	14	0.0	NaN
533	0	NaN	0	0	0.0	0	0	0.0	NaN
5842	4	NaN	9	10	0.0	1	2	0.0	NaN

[7255 rows x 22 columns]

```
[1002]: df[(df['CS']).isnull()].apply(lambda x: x['SB'] / 2, axis=1)
```

```
[1002]: 14448    0.0
16285    0.0
15712    0.0
15131    0.0
16286    0.5
...
532      1.0
4782     4.5
5312     6.5
533      0.0
5842     2.0
Length: 7255, dtype: float64
```

```
[1003]: df[(df['CS']).isnull()].apply(lambda x: x['SB'] / 2, axis=1).value_counts()
```

```
[1003]: 0.0    4494
0.5     794
1.0     438
1.5     303
2.0     257
2.5     165
3.0     139
3.5     131
4.0      91
4.5      81
5.0      51
```

```

5.5      50
6.5      38
6.0      36
7.5      28
7.0      22
8.0      21
9.0      19
8.5      16
9.5      11
11.5     8
10.0     8
10.5     8
11.0     7
13.0     6
14.0     5
12.0     5
14.5     3
18.5     3
12.5     2
17.5     2
13.5     2
16.5     2
16.0     2
20.0     1
15.0     1
15.5     1
24.0     1
18.0     1
17.0     1
21.5     1
dtype: int64

```

I don't love the max of 24, but overall these values look good and we definitely don't have many of the higher values. So we're going to apply this to our missing CS data

First I'm going to test it out on a copy

```
[1004]: df_temp = df[(df['CS']).isnull()]
```

```
[1005]: df_temp['CS'] = df_temp.apply(lambda x: x['SB'] / 2, axis=1)
```

```
[1006]: df_temp[(df_temp['retroID'] == 'zitzmbi01')]
```

```
[1006]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
4241	zitzmbi01	1926	1	CIN	NL	53	94	21	23	2	...	3	
532	zitzmbi01	1919	1	PIT	NL	11	26	5	5	1	...	2	
4782	zitzmbi01	1927	1	CIN	NL	88	232	47	66	10	...	24	
5312	zitzmbi01	1928	1	CIN	NL	101	266	53	79	9	...	33	

533	zitzmbi01	1919	2	CIN	NL	2	1	0	0	0	...	0
5842	zitzmbi01	1929	1	CIN	NL	47	84	18	19	3	...	6

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
4241	3	1.5	6	7	0.0	2	3	0.0	NaN
532	2	1.0	0	6	0.0	0	1	0.0	NaN
4782	9	4.5	20	18	0.0	4	17	0.0	NaN
5312	13	6.5	13	22	0.0	3	14	0.0	NaN
533	0	0.0	0	0	0.0	0	0	0.0	NaN
5842	4	2.0	9	10	0.0	1	2	0.0	NaN

[6 rows x 22 columns]

We know from before that this guy had NaNs for his CS and now it's all filled in, so our plan worked. Let's do it for the actual data

I don't know how to reassign values to a subset of a DataFrame based on a predicate (or if it's possible), so we'll get a little hacky and apply a function with a conditional. Here's what I tried originally:

```
df[(df['CS']).isnull()]['CS'] = df.apply(lambda x: x['SB'] / 2, axis=1)
```

```
[1007]: def fill_cs(data):
        if math.isnan(data['CS']):
            return data['SB'] / 2
        else:
            return data['CS']
```

```
[1008]: df['CS'] = df.apply(lambda x: fill_cs(x), axis=1)
```

```
[1009]: df[(df['retroID'] == 'zitzmbi01')]
```

```
[1009]:
```

	retroID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	\
4241	zitzmbi01	1926	1	CIN	NL	53	94	21	23	2	...	3	
532	zitzmbi01	1919	1	PIT	NL	11	26	5	5	1	...	2	
3715	zitzmbi01	1925	1	CIN	NL	104	301	53	76	13	...	21	
4782	zitzmbi01	1927	1	CIN	NL	88	232	47	66	10	...	24	
5312	zitzmbi01	1928	1	CIN	NL	101	266	53	79	9	...	33	
533	zitzmbi01	1919	2	CIN	NL	2	1	0	0	0	...	0	
5842	zitzmbi01	1929	1	CIN	NL	47	84	18	19	3	...	6	

  

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
4241	3	1.5	6	7	0.0	2	3	0.0	NaN
532	2	1.0	0	6	0.0	0	1	0.0	NaN
3715	11	11.0	35	22	0.0	6	2	0.0	NaN
4782	9	4.5	20	18	0.0	4	17	0.0	NaN
5312	13	6.5	13	22	0.0	3	14	0.0	NaN
533	0	0.0	0	0	0.0	0	0	0.0	NaN

```
5842    4    2.0    9   10   0.0    1    2   0.0   NaN
```

```
[7 rows x 22 columns]
```

```
[1010]: 100 * df.isnull().sum() / len(df)
```

```
[1010]: retroID    0.000000
yearID    0.000000
stint     0.000000
teamID    0.000000
lgID      0.000000
G         0.000000
AB        0.000000
R         0.000000
H         0.000000
2B        0.000000
3B        0.000000
HR        0.000000
RBI       0.000000
SB        0.000000
CS        0.000000
BB        0.000000
SO        0.000000
IBB       0.000000
HBP       0.000000
SH        0.000000
SF        0.000000
GIDP      9.839985
dtype: float64
```

Handling missing GIDP data

```
[1011]: df[(df['GIDP'].isnull())]
```

```
[1011]:      retroID  yearID  stint  teamID  lgID    G  AB  R  H  2B  ...  RBI  \
2082  abramge01   1923     1    CIN   NL     3   1  0  1  0  ...    0
534   acostjo01   1920     1    WS1   AL    17  25  2  6  1  ...    1
1569  acostjo01   1922     1    CHA   AL     5   5  0  1  0  ...    0
1049  acostjo01   1921     1    WS1   AL    33  30  2  2  0  ...    0
6374  adairji01   1931     1    CHN   NL    18  76  9  21  3  ...    3
...      ...      ...    ...    ...   ...  ...  ...  ...  ...  ...  ...
5312  zitzmbi01   1928     1    CIN   NL   101  266  53  79  9  ...   33
533   zitzmbi01   1919     2    CIN   NL     2   1  0  0  0  ...    0
5842  zitzmbi01   1929     1    CIN   NL    47   84  18  19  3  ...    6
9445  zuberbi01   1936     1    CLE   AL     2   5  1  1  0  ...    0
10501 zuberbi01   1938     1    CLE   AL    15   7  0  0  0  ...    0
```

	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
2082	0	0.0	0	0	0.0	0	0	0.0	NaN
534	0	0.0	4	7	0.0	0	2	0.0	NaN
1569	0	0.0	1	1	0.0	0	0	0.0	NaN
1049	1	0.0	6	14	0.0	0	1	0.0	NaN
6374	1	0.5	1	8	0.0	0	2	0.0	NaN
...	..	...	..	..	...	...	..	...	...
5312	13	6.5	13	22	0.0	3	14	0.0	NaN
533	0	0.0	0	0	0.0	0	0	0.0	NaN
5842	4	2.0	9	10	0.0	1	2	0.0	NaN
9445	0	0.0	0	1	0.0	0	0	0.0	NaN
10501	0	0.0	0	1	0.0	0	1	0.0	NaN

[8683 rows x 22 columns]

```
[1012]: df[(df['GIDP'].isnull())]['yearID'].max()
```

```
[1012]: 1938
```

```
[1013]: df[(df['yearID'] < 1939)].shape[0]
```

```
[1013]: 10502
```

```
[1014]: df[(df['GIDP'].isnull())].shape[0]
```

```
[1014]: 8683
```

```
[1015]: 8683/10502
```

```
[1015]: 0.8267948962102457
```

Over 82% of records before 1939 are missing GIDP, but it doesn't extend beyond that. I think we can once again just fill the values in with 0

```
[1016]: df['GIDP'].fillna(value=0, inplace=True)
```

```
[1017]: 100 * df.isnull().sum() / len(df)
```

```
[1017]: retroID    0.0
yearID       0.0
stint        0.0
teamID       0.0
lgID         0.0
G            0.0
AB           0.0
R            0.0
H            0.0
```

```

2B          0.0
3B          0.0
HR          0.0
RBI         0.0
SB          0.0
CS          0.0
BB          0.0
SO          0.0
IBB         0.0
HBP         0.0
SH          0.0
SF          0.0
GIDP        0.0
dtype: float64

```

We've handled all missing data in the batting database

### Data Integration

Now we need to eliminate an columns that we don't want (if any) and convert the ones we keep to numerical values.

```
[1018]: df.head()
```

```

[1018]:      retroID  yearID  stint teamID lgID   G  AB  R  H  2B  ...  RBI  SB  \
79400  aardsda01   2013      1    NYN   NL  43   0  0  0   0  ...   0   0
82244  aardsda01   2015      1    ATL   NL  33   1  0  0   0  ...   0   0
69712  aardsda01   2006      1    CHN   NL  45   2  0  0   0  ...   0   0
73859  aardsda01   2009      1    SEA   AL  73   0  0  0   0  ...   0   0
71089  aardsda01   2007      1    CHA   AL  25   0  0  0   0  ...   0   0

      CS  BB  SO  IBB  HBP  SH   SF  GIDP
79400  0.0  0   0  0.0   0   0  0.0  0.0
82244  0.0  0   1  0.0   0   0  0.0  0.0
69712  0.0  0   0  0.0   0   1  0.0  0.0
73859  0.0  0   0  0.0   0   0  0.0  0.0
71089  0.0  0   0  0.0   0   0  0.0  0.0

```

```
[5 rows x 22 columns]
```

```
[1019]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 88242 entries, 79400 to 86706
Data columns (total 22 columns):
retroID      88242 non-null object
yearID       88242 non-null int64
stint        88242 non-null int64
teamID       88242 non-null object

```

```

lgID      88242 non-null object
G         88242 non-null int64
AB        88242 non-null int64
R         88242 non-null int64
H         88242 non-null int64
2B        88242 non-null int64
3B        88242 non-null int64
HR        88242 non-null int64
RBI       88242 non-null int64
SB        88242 non-null int64
CS        88242 non-null float64
BB        88242 non-null int64
SO        88242 non-null int64
IBB       88242 non-null float64
HBP       88242 non-null int64
SH        88242 non-null int64
SF        88242 non-null float64
GIDP      88242 non-null float64
dtypes: float64(4), int64(15), object(3)
memory usage: 15.5+ MB

```

We will handle the metadata columns later and only worry about numerical columns for now

```
[1020]: df['lgID'].value_counts()
```

```

[1020]: NL      44129
        AL      44113
        Name: lgID, dtype: int64

```

```
[1021]: pd.get_dummies(df['lgID'], drop_first=True)
```

```

[1021]:      NL
79400    1
82244    1
69712    1
73859    0
71089    0
...     ..
20499    0
18050    0
83729    0
85212    0
86706    0

```

```
[88242 rows x 1 columns]
```

This one will be easy - there are only two leagues in the dataset, so we can just transform that into a single boolean column. Of course that column will be NL, the superior league.



```
[1022]: df['NL'] = pd.get_dummies(df['lgID'], drop_first=True)
df.drop(columns=['lgID'], inplace=True)
```

```
[1023]: df
```

```
[1023]:
```

	retroID	yearID	stint	teamID	G	AB	R	H	2B	3B	...	SB	CS	\
79400	aardsda01	2013	1	NYN	43	0	0	0	0	0	...	0	0.0	
82244	aardsda01	2015	1	ATL	33	1	0	0	0	0	...	0	0.0	
69712	aardsda01	2006	1	CHN	45	2	0	0	0	0	...	0	0.0	
73859	aardsda01	2009	1	SEA	73	0	0	0	0	0	...	0	0.0	
71089	aardsda01	2007	1	CHA	25	0	0	0	0	0	...	0	0.0	
...	...	...	...	...	..	..	..	..	..	..	...	..	...	
20499	zuverge01	1955	2	BAL	28	23	1	5	1	0	...	0	0.0	
18050	zuverge01	1951	1	CLE	16	0	0	0	0	0	...	0	0.0	
83729	zychto01	2015	1	SEA	13	0	0	0	0	0	...	0	0.0	
85212	zychto01	2016	1	SEA	12	0	0	0	0	0	...	0	0.0	
86706	zychto01	2017	1	SEA	45	0	0	0	0	0	...	0	0.0	

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
79400	0	0	0.0	0	0	0.0	0.0	1
82244	0	1	0.0	0	0	0.0	0.0	1
69712	0	0	0.0	0	1	0.0	0.0	1
73859	0	0	0.0	0	0	0.0	0.0	0
71089	0	0	0.0	0	0	0.0	0.0	0
...	..	..	...	...	..	...	...	..
20499	1	5	0.0	0	1	0.0	1.0	0
18050	0	0	0.0	0	0	0.0	0.0	0
83729	0	0	0.0	0	0	0.0	0.0	0
85212	0	0	0.0	0	0	0.0	0.0	0
86706	0	0	0.0	0	0	0.0	0.0	0

[88242 rows x 22 columns]

Now we need to figure out how to handle the teamID column.

```
[1024]: df['teamID'].nunique()
```

```
[1024]: 45
```

Since we have more than 30 team IDs, to keep things consistent I'm just going to map them to franchise ID.

```
[1025]: from team_dict import team_dict, team_nums
df['teamID'] = df['teamID'].apply(lambda x: team_dict()[x])
```

Now let's look at what other data integration tasks we need to do

```
[1026]: df['teamID'].nunique()
```

[1026]: 30

We're now all set with team IDs as strings

```
[1027]: df.head()
```

```
[1027]:
```

	retroID	yearID	stint	teamID	G	AB	R	H	2B	3B	...	SB	CS	\
79400	aardsda01	2013	1	NYM	43	0	0	0	0	0	...	0	0.0	
82244	aardsda01	2015	1	ATL	33	1	0	0	0	0	...	0	0.0	
69712	aardsda01	2006	1	CHC	45	2	0	0	0	0	...	0	0.0	
73859	aardsda01	2009	1	SEA	73	0	0	0	0	0	...	0	0.0	
71089	aardsda01	2007	1	CHW	25	0	0	0	0	0	...	0	0.0	

  

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
79400	0	0	0.0	0	0	0.0	0.0	1
82244	0	1	0.0	0	0	0.0	0.0	1
69712	0	0	0.0	0	1	0.0	0.0	1
73859	0	0	0.0	0	0	0.0	0.0	0
71089	0	0	0.0	0	0	0.0	0.0	0

[5 rows x 22 columns]

```
[1028]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88242 entries, 79400 to 86706
Data columns (total 22 columns):
retroID      88242 non-null object
yearID       88242 non-null int64
stint        88242 non-null int64
teamID       88242 non-null object
G            88242 non-null int64
AB           88242 non-null int64
R            88242 non-null int64
H            88242 non-null int64
2B           88242 non-null int64
3B           88242 non-null int64
HR           88242 non-null int64
RBI          88242 non-null int64
SB           88242 non-null int64
CS           88242 non-null float64
BB           88242 non-null int64
SO           88242 non-null int64
IBB          88242 non-null float64
HBP          88242 non-null int64
SH           88242 non-null int64
SF           88242 non-null float64
GIDP         88242 non-null float64
```

```

NL          88242 non-null uint8
dtypes: float64(4), int64(15), object(2), uint8(1)
memory usage: 14.9+ MB

```

```
[1029]: df = df.sort_index()
```

```
[1030]: df.head()
```

```
[1030]:
```

	retroID	yearID	stint	teamID	G	AB	R	H	2B	3B	...	SB	CS	\
0	adamsba01	1919	1	PIT	34	92	2	17	2	1	...	0	0.0	
1	adamsbe01	1919	1	PHI	78	232	14	54	7	2	...	4	2.0	
2	adamswi01	1919	1	OAK	1	2	0	0	0	0	...	0	0.0	
3	agnewsa01	1919	1	MIN	42	98	6	23	7	0	...	1	0.5	
4	ainsmed01	1919	1	DET	114	364	42	99	17	12	...	9	4.5	

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	6	13	0.0	0	3	0.0	0.0	1
1	6	27	0.0	0	3	0.0	0.0	1
2	0	1	0.0	0	0	0.0	0.0	0
3	10	8	0.0	1	9	0.0	0.0	0
4	45	30	0.0	1	12	0.0	0.0	0

```
[5 rows x 22 columns]
```

We need some sort of dictionary to associate a player's retroID with an index. The following steps care of that. This is so we can later associate the correct retroID with our data.

```
[1031]: df.reset_index(inplace=True)
```

```
[1032]: metadata_column_labels = ['index', 'yearID', 'stint', 'teamID']
```

```
[1033]: metadata = df[metadata_column_labels].set_index(df['retroID']).reset_index()
```

```
[1034]: metadata.head()
```

```
[1034]:
```

	retroID	index	yearID	stint	teamID
0	adamsba01	0	1919	1	PIT
1	adamsbe01	1	1919	1	PHI
2	adamswi01	2	1919	1	OAK
3	agnewsa01	3	1919	1	MIN
4	ainsmed01	4	1919	1	DET

The metadata table will eventually be expanded with information from Players.csv to hold all relevant player information that isn't used for the neural network.

```
[1035]: indexer = metadata.drop_duplicates('retroID').set_index('index').T.  
         ↳to_dict('retroID')[0]
```

```
[1036]: df = df.drop(columns=metadata_column_labels)
```

```
[1037]: df.head()
```

```
[1037]:
```

	retroID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	\
0	adamsba01	34	92	2	17	2	1	0	4	0	0.0	6	13	0.0	0	
1	adamsbe01	78	232	14	54	7	2	1	17	4	2.0	6	27	0.0	0	
2	adamswi01	1	2	0	0	0	0	0	0	0	0.0	0	1	0.0	0	
3	agnewsa01	42	98	6	23	7	0	0	10	1	0.5	10	8	0.0	1	
4	ainsmed01	114	364	42	99	17	12	3	32	9	4.5	45	30	0.0	1	

  

	SH	SF	GIDP	NL
0	3	0.0	0.0	1
1	3	0.0	0.0	1
2	0	0.0	0.0	0
3	9	0.0	0.0	0
4	12	0.0	0.0	0

Now that the metadata is gone, we just have the ID and the numerical batting information. We can group by the ID and just sum every other column to get player career totals.

```
[1038]: df = df.groupby('retroID').sum().reset_index()
```

```
[1039]: df
```

```
[1039]:
```

	retroID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	\
0	aardsda01	331	4	0	0	0	0	0	0	0	0.0	
1	aaronha01	3298	12364	2174	3771	624	98	755	2297	240	73.0	
2	aaronto01	437	944	102	216	42	6	13	94	9	8.0	
3	aasedo01	448	5	0	0	0	0	0	0	0	0.0	
4	abadan01	15	21	1	2	0	0	0	0	0	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	
15187	zupcibo01	319	795	99	199	47	4	7	80	7	5.0	
15188	zupofr01	16	18	3	3	1	0	0	0	0	0.0	
15189	zuvelpa01	209	491	41	109	17	2	2	20	2	0.0	
15190	zuverge01	266	142	5	21	2	1	0	7	0	1.0	
15191	zychto01	70	0	0	0	0	0	0	0	0	0.0	

  

	BB	SO	IBB	HBP	SH	SF	GIDP	NL
0	0	2	0.0	0	1	0.0	0.0	4
1	1402	1383	293.0	32	21	121.0	328.0	21
2	86	145	3.0	0	9	6.0	36.0	7
3	0	3	0.0	0	0	0.0	0.0	2
4	4	5	0.0	0	0	0.0	1.0	1
...	...	...	...	...	...	...	...	...
15187	57	137	3.0	6	20	8.0	15.0	0
15188	2	6	0.0	0	0	0.0	0.0	0

15189	34	50	1.0	2	18	0.0	8.0	4
15190	9	39	0.0	0	16	0.0	3.0	1
15191	0	0	0.0	0	0	0.0	0.0	0

[15192 rows x 19 columns]

Since we summed everything, we just need to change the NL column back. We can divide each value by itself to get either 1 or 0 like we had before.

```
[1046]: df['NL'] = np.where(df['NL'] > 0, 1, 0)
```

```
[1047]: tensor = df.drop(columns=['retroID'])
```

```
[1049]: tensor
```

```
[1049]:
```

	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO \
0	331	4	0	0	0	0	0	0	0	0.0	0	2
1	3298	12364	2174	3771	624	98	755	2297	240	73.0	1402	1383
2	437	944	102	216	42	6	13	94	9	8.0	86	145
3	448	5	0	0	0	0	0	0	0	0.0	0	3
4	15	21	1	2	0	0	0	0	0	1.0	4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
15187	319	795	99	199	47	4	7	80	7	5.0	57	137
15188	16	18	3	3	1	0	0	0	0	0.0	2	6
15189	209	491	41	109	17	2	2	20	2	0.0	34	50
15190	266	142	5	21	2	1	0	7	0	1.0	9	39
15191	70	0	0	0	0	0	0	0	0	0.0	0	0

	IBB	HBP	SH	SF	GIDP	NL
0	0.0	0	1	0.0	0.0	1
1	293.0	32	21	121.0	328.0	1
2	3.0	0	9	6.0	36.0	1
3	0.0	0	0	0.0	0.0	1
4	0.0	0	0	0.0	1.0	1
...	...	...	...	...	...	...
15187	3.0	6	20	8.0	15.0	0
15188	0.0	0	0	0.0	0.0	0
15189	1.0	2	18	0.0	8.0	1
15190	0.0	0	16	0.0	3.0	1
15191	0.0	0	0	0.0	0.0	0

[15192 rows x 18 columns]

We now have a tensor with only relevant information, an indexing dictionary to get the player for each row, and a (soon to be expanded) metadata table to get more information on each player.

```
[ ]:
```