fielding_pre

March 9, 2020

[47]:
```
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
```

[48]:
```
df = pd.read_csv('../data/lahman/mlb_data/Fielding.csv').sort_values('playerID')
```

[49]:
```
# This will be exported to a separate module
ids = pd.read_csv('../data/lahman/mlb_data/People.csv')
ids = ids[['playerID', 'retroID']]
id_dict = ids.set_index('playerID').to_dict()['retroID']

def get_retroid(id):
    return id_dict[id] if id_dict is not None else id
```

[50]:
```
df['playerID'] = df['playerID'].apply(get_retroid)
df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

Exploration

[51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 112837 entries, 85308 to 106797
Data columns (total 18 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   retroID  112837 non-null  object 
 1   yearID   112837 non-null  int64  
 2   stint    112837 non-null  int64  
 3   teamID   112837 non-null  object 
 4   lgID     112837 non-null  object 
 5   POS      112837 non-null  object 
 6   G        112837 non-null  int64  
 7   GS       89431 non-null   float64
 8   InnOuts  89431 non-null   float64
 9   PO       112837 non-null  int64  
 10  A        112837 non-null  int64  
 11  E        112836 non-null  float64
```

```
 12  DP        112837 non-null  int64
 13  PB        8538 non-null    float64
 14  WP        1169 non-null    float64
 15  SB        6389 non-null    float64
 16  CS        6389 non-null    float64
 17  ZR        1169 non-null    float64
dtypes: float64(8), int64(6), object(4)
memory usage: 16.4+ MB
```

[52]:
```
df.shape
```

[52]:
```
(112837, 18)
```

[53]:
```
df.columns
```

[53]:
```
Index(['retroID', 'yearID', 'stint', 'teamID', 'lgID', 'POS', 'G', 'GS',
       'InnOuts', 'PO', 'A', 'E', 'DP', 'PB', 'WP', 'SB', 'CS', 'ZR'],
      dtype='object')
```

We want to get rid of columns which already exist in the Batting DataFrame (with which we will be merging this)

[54]:
```
columns_to_drop = ['stint', 'teamID', 'lgID', 'G']
```

[55]:
```
df.drop(columns=columns_to_drop, inplace=True)
```

[56]:
```
df.head()
```

[56]:

	retroID	yearID	POS	GS	InnOuts	PO	A	E	DP	PB	WP	SB	CS	ZR
85308	aardd001	2004	P	0.0	32.0	0	0	0.0	0	NaN	NaN	NaN	NaN	NaN
101187	aardd001	2013	P	0.0	119.0	1	5	0.0	0	NaN	NaN	NaN	NaN	NaN
99344	aardd001	2012	P	0.0	3.0	0	0	0.0	0	NaN	NaN	NaN	NaN	NaN
95793	aardd001	2010	P	0.0	149.0	2	3	1.0	0	NaN	NaN	NaN	NaN	NaN
104866	aardd001	2015	P	0.0	92.0	0	1	1.0	0	NaN	NaN	NaN	NaN	NaN

Cleaning and Preprocessing

We see a lot of NaNs in the last 5 columns. According to the Lahman readme, these are:

- PB - Passed Balls (by catchers)
- WP - Wild Pitches (by catchers)
- SB - Opponent Stolen Bases (by catchers)
- CS - Opponents Caught Stealing (by catchers)
- ZR - Zone Rating

It looks like the data demands that we treat catchers separately from other position players. This intuitively makes sense from what we know about baseball, and it saves us from getting rid of a lot of data. First, though, let's look at how much of that data is missing if we JUST look at catchers.

[57]:
```
df_catchers = df[df['POS'] == 'C']
```

[58]:
```
# Get missing data in the catchers category as a percentage
100 * df_catchers.isnull().sum() / len(df)
```

[58]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         1.901858
InnOuts    1.901858
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         6.530659
SB         1.904517
CS         1.904517
ZR         6.530659
dtype: float64
```

Most of the percentages are negligable, but we can take a look at WP and ZR and see if the missing data is from early years.

[59]:
```
early_catchers = df_catchers[df_catchers['yearID'] < 1955]
```

[60]:
```
100 * early_catchers.isnull().sum() / len(df)
```

[60]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         1.901858
InnOuts    1.901858
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         1.901858
SB         1.901858
CS         1.901858
ZR         1.901858
dtype: float64
```

Definitely not the case. Let's try to narrow down where the issue is.

[61]:
```
post1985_catchers = df_catchers[df_catchers['yearID'] > 1985]
```

[62]:
```
100 * post1985_catchers.isnull().sum() / len(df)
```

[62]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         3.265773
SB         0.000000
CS         0.000000
ZR         3.265773
dtype: float64
```

[63]:
```
df_1955_to_1986_catchers = df_catchers[(df_catchers['yearID'] >= 1955) &␣
↪(df_catchers['yearID'] <= 1985)]
```

[64]:
```
100 * df_1955_to_1986_catchers.isnull().sum() / len(df)
```

[64]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         1.363028
SB         0.002659
CS         0.002659
ZR         1.363028
dtype: float64
```

[65]:
```
pre_1930_catchers = df_catchers[df_catchers['yearID'] < 1930]
```

[66]:
```
100 * pre_1930_catchers.isnull().sum() / len(df)
```

[66]:
```
retroID    0.000000
yearID     0.000000
POS        0.000000
GS         0.591118
```

```
InnOuts    0.591118
PO         0.000000
A          0.000000
E          0.000000
DP         0.000000
PB         0.000000
WP         0.591118
SB         0.591118
CS         0.591118
ZR         0.591118
dtype: float64
```

We see that the issue is mainly in the very early years, and we are fine with dropping that information by just filling it in as we did in the Batters table.

So with that, we are fine with filling all NA values with 0.

[67]:
```
df_catchers['GS'].fillna(value=0, inplace=True)
df_catchers['InnOuts'].fillna(value=0, inplace=True)
df_catchers['WP'].fillna(value=0, inplace=True)
df_catchers['SB'].fillna(value=0, inplace=True)
df_catchers['CS'].fillna(value=0, inplace=True)
df_catchers['ZR'].fillna(value=0, inplace=True)
```

[68]:
```
df['GS'].fillna(value=0, inplace=True)
df['InnOuts'].fillna(value=0, inplace=True)
#We can just drop the catcher-related columns from the original dataframe, as we␣
 ↪will also drop all catcher rows
catcher_columns = ['PB', 'WP', 'SB', 'CS', 'ZR']
df.drop(columns=catcher_columns, inplace=True)
```

Now drop all catcher rows so we have two separate dataframes, and get rid of the yearID column which we're done with and will be useless after aggregation.

[69]:
```
df = df[df['POS'] != 'C']
```

[70]:
```
df.drop(columns=['yearID'], inplace=True)
df_catchers.drop(columns=['yearID'], inplace=True)
```

[71]:
```
df.shape
```

[71]:
```
(104299, 8)
```

[72]:
```
df_catchers.shape
```

[72]:
```
(8538, 13)
```

[73]:
```
100 * df.isnull().sum() / len(df)
```

[73]:
```
retroID    0.000000
POS        0.000000
GS         0.000000
InnOuts    0.000000
PO         0.000000
A          0.000000
E          0.000959
DP         0.000000
dtype: float64
```

Now we just see a little bit of information missing from Errors, so we can fill that with 0s no problem.

[74]:
```
df['E'].fillna(value=0, inplace=True)
```

[75]:
```
100 * df.isnull().sum() / len(df)
```

[75]:
```
retroID    0.0
POS        0.0
GS         0.0
InnOuts    0.0
PO         0.0
A          0.0
E          0.0
DP         0.0
dtype: float64
```

[76]:
```
100 * df_catchers.isnull().sum() / len(df)
```

[76]:
```
retroID    0.0
POS        0.0
GS         0.0
InnOuts    0.0
PO         0.0
A          0.0
E          0.0
DP         0.0
PB         0.0
WP         0.0
SB         0.0
CS         0.0
ZR         0.0
dtype: float64
```

Aggregation

Now we just need to aggregate all stats to get total career numbers for each player.

[77]:
```
df = df.groupby('retroID').sum().reset_index()
```

[78]:
```
df_catchers = df_catchers.groupby('retroID').sum().reset_index()
```

[79]:
```
df
```

[79]:

	retroID	GS	InnOuts	PO	A	E	DP
0	aardd001	0.0	1011.0	11	29	3.0	2
1	aaroh101	2977.0	78414.0	7436	429	144.0	218
2	aarot101	206.0	6472.0	1317	113	22.0	124
3	aased001	91.0	3328.0	67	135	13.0	10
4	abada001	4.0	138.0	37	1	1.0	3
...	...	...	...	...	...	...	...
14222	zumaj001	0.0	629.0	7	14	2.0	1
14223	zupcb001	198.0	5842.0	483	22	12.0	5
14224	zuveg101	31.0	1847.0	45	145	7.0	10
14225	zuvep001	136.0	3844.0	267	415	23.0	84
14226	zycht001	1.0	218.0	1	6	1.0	0

[14227 rows x 7 columns]

[80]:
```
df_catchers
```

[80]:

	retroID	GS	InnOuts	PO	A	E	DP	PB	WP	SB	CS	\
0	adamb105	1.0	27.0	6	0	0.0	0	0.0	0.0	1.0	0.0	
1	adamb106	0.0	0.0	249	90	12.0	15	7.0	0.0	0.0	0.0	
2	adamd101	3.0	78.0	9	2	0.0	0	1.0	0.0	0.0	0.0	
3	adled101	65.0	1840.0	453	26	4.0	2	8.0	19.0	37.0	16.0	
4	afent001	20.0	613.0	123	5	1.0	3	6.0	0.0	17.0	3.0	
...	...	...	...	...	...	...	..	...	...	...	...	
1524	zimmd101	27.0	744.0	150	18	6.0	1	5.0	12.0	10.0	10.0	
1525	zimmj101	298.0	8560.0	2131	150	21.0	26	19.0	84.0	110.0	80.0	
1526	zinta001	0.0	3.0	2	0	0.0	0	0.0	0.0	0.0	0.0	
1527	zunim001	535.0	14489.0	4356	264	21.0	22	39.0	0.0	248.0	98.0	
1528	zupof101	1.0	114.0	31	1	2.0	0	1.0	1.0	2.0	1.0	

	ZR
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
1524	3.0
1525	4.0
1526	0.0
1527	0.0

```
1528  0.0

[1529 rows x 12 columns]
```

[ ]: