pitching_pre

April 28, 2020

[211]:
```
import pandas as pd
import numpy as np
pd.options.mode.chained_assignment = None  # default='warn'
```

[212]:
```
df = pd.read_csv('../data/lahman/mlb_data/Pitching.csv')
```

[213]:
```
# 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
```

[214]:
```
df['playerID'] = df['playerID'].apply(get_retroid)
df.rename(columns={'playerID': 'retroID'}, inplace=True)
```

Exploration

[215]:
```
df.head()
```

[215]:

	retroID	yearID	stint	teamID	lgID	W	L	G	GS	CG	...	IBB	WP	HBP	BK	BFP	GF	R	SH	SF	GIDP
0	adamb104	1919	1	PIT	NL	17	10	34	29	23	...	NaN	2	3	0	1017.0	5	66	NaN	NaN	NaN
1	adamw101	1919	1	PHA	AL	0	0	1	0	0	...	NaN	0	1	0	21.0	1	2	NaN	NaN	NaN
2	alexg102	1919	1	CHN	NL	16	11	30	27	20	...	NaN	1	0	0	906.0	3	51	NaN	NaN	NaN
3	altrn101	1919	1	WS1	AL	0	0	1	0	0	...	NaN	0	0	0	4.0	0	4	NaN	NaN	NaN
4	amesr101	1919	1	SLN	NL	3	5	23	7	1	...	NaN	3	1	0	314.0	10	44	NaN	NaN	NaN

[5 rows x 30 columns]

[216]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40372 entries, 0 to 40371
Data columns (total 30 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   retroID  40372 non-null  object 
 1   yearID   40372 non-null  int64  
 2   stint    40372 non-null  int64  
 3   teamID   40372 non-null  object 
 4   lgID     40372 non-null  object 
 5   W        40372 non-null  int64  
 6   L        40372 non-null  int64  
 7   G        40372 non-null  int64  
 8   GS       40372 non-null  int64  
 9   CG       40372 non-null  int64  
 10  SHO      40372 non-null  int64  
 11  SV       40372 non-null  int64  
 12  IPouts   40372 non-null  int64  
 13  H        40372 non-null  int64  
 14  ER       40372 non-null  int64  
 15  HR       40372 non-null  int64  
 16  BB       40372 non-null  int64  
 17  SO       40372 non-null  int64  
 18  BAOpp    40360 non-null  float64
 19  ERA      40298 non-null  float64
 20  IBB      32121 non-null  float64
 21  WP       40372 non-null  int64  
 22  HBP      40372 non-null  int64  
 23  BK       40372 non-null  int64  
 24  BFP      40369 non-null  float64
 25  GF       40372 non-null  int64  
 26  R        40372 non-null  int64  
 27  SH       27512 non-null  float64
 28  SF       27512 non-null  float64
 29  GIDP     26381 non-null  float64
dtypes: float64(7), int64(20), object(3)
memory usage: 9.2+ MB
```

[217]:
```
df.columns
```

[217]:
```
Index(['retroID', 'yearID', 'stint', 'teamID', 'lgID', 'W', 'L', 'G', 'GS',
       'CG', 'SHO', 'SV', 'IPouts', 'H', 'ER', 'HR', 'BB', 'SO', 'BAOpp',
       'ERA', 'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF', 'R', 'SH', 'SF', 'GIDP'],
      dtype='object')
```

[218]:
```
columns_to_drop = ['stint', 'teamID', 'lgID']
```

[219]:
```
df.drop(columns=columns_to_drop, inplace=True)
```

[220]:
```
df.shape
```

[220]:
```
(40372, 27)
```

Cleaning and Preprocessing

[221]:
```
100 * df.isnull().sum() / len(df)
```

[221]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.029724
ERA         0.183295
IBB        20.437432
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH         31.853760
SF         31.853760
GIDP       34.655207
dtype: float64
```

[222]:
```
df_early = df[df['yearID'] <= 1930]
```

[223]:
```
100 * df_early.isnull().sum() / len(df)
```

[223]:
```
retroID    0.000000
yearID     0.000000
```

```
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.002477
ERA         0.042108
IBB         6.442584
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH          6.442584
SF          6.442584
GIDP        6.442584
dtype: float64
```

[224]:
```
df_modern = df[df['yearID'] >= 1980]
```

[225]:
```
100 * df_modern.isnull().sum() / len(df)
```

[225]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.019816
```

```
ERA        0.056970
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.000000
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

Luckily the more modern data is barely missing any information.

[226]:
```
df_mid = df[(df['yearID'] > 1935) & (df['yearID'] < 1975)]
```

[227]:
```
100 * df_mid.isnull().sum() / len(df)
```

[227]:
```
retroID     0.000000
yearID      0.000000
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.007431
ERA         0.066878
IBB        11.428713
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.000000
GF          0.000000
R           0.000000
SH         22.845041
SF         22.845041
GIDP       25.646488
dtype: float64
```

We see that much of the lost data comes within this 40-year span. I think that given what the major missing information is - intentional bases on balls, sacrifice hits, sacrifice flies and grounded into double play - and the fact that these statistics are not often used as primary indicators of a pitcher's ability, coupled with the fact that it's mostly localized within less than half of our time frame, I can be forgiven for just filling these values as 0.

[228]:
```
df['IBB'].fillna(0, inplace=True)
df['SH'].fillna(0, inplace=True)
df['SF'].fillna(0, inplace=True)
df['GIDP'].fillna(0, inplace=True)
```

[229]:
```
100 * df.isnull().sum() / len(df)
```

[229]:
```
retroID    0.000000
yearID     0.000000
W          0.000000
L          0.000000
G          0.000000
GS         0.000000
CG         0.000000
SHO        0.000000
SV         0.000000
IPouts     0.000000
H          0.000000
ER         0.000000
HR         0.000000
BB         0.000000
SO         0.000000
BAOpp      0.029724
ERA        0.183295
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.007431
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

We're left with three fields that having missing data: opponents' batting average, earned run average and batters faced by pitcher. We'll have to do some data exploration on these because I don't want to just fill them with 0s.

Missing Values: BAOpp

[230]:
```
df_baopp_missing = df[df['BAOpp'].isnull()]
```

[231]:
```
df_baopp_missing.shape
```

[231]:
```
(12, 27)
```

[232]:
```
df_baopp_missing.sort_values('retroID')
```

[232]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
14000	apodb101	1973	0	0	1	0	0	0	0	0	...	0.0	0	0	
19114	arrof001	1986	0	0	1	0	0	0	0	0	...	0.0	0	0	
39581	brotr001	2018	0	0	1	0	0	0	0	0	...	0.0	0	0	
36447	dunnj001	2014	0	0	1	0	0	0	0	2	...	0.0	2	0	
38848	eschj001	2017	0	0	1	0	0	0	0	0	...	0.0	0	0	
3709	fordw103	1936	0	0	1	0	0	0	0	0	...	0.0	0	0	
297	glasn101	1920	0	0	1	0	0	0	0	7	...	0.0	0	1	
26791	halts001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
27036	radis001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
36189	tolls002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
36208	villb002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
13621	youn1101	1971	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
14000	0	2.0	0	1	0.0	0.0	0.0
19114	0	3.0	0	0	0.0	0.0	0.0
39581	0	2.0	0	1	0.0	0.0	0.0
36447	0	2.0	0	0	0.0	1.0	0.0
38848	0	2.0	0	0	0.0	0.0	0.0
3709	0	3.0	0	2	0.0	0.0	0.0
297	0	12.0	0	4	0.0	0.0	0.0
26791	0	1.0	0	0	0.0	0.0	0.0
27036	0	1.0	0	0	0.0	0.0	0.0
36189	0	2.0	0	0	0.0	0.0	0.0
36208	0	1.0	1	0	0.0	0.0	0.0
13621	0	0.0	0	0	0.0	0.0	0.0

[12 rows x 27 columns]

Nobody appears in this table more than once. We'll hope that we can get career numbers for them and fill with the average. For anyone who only appears once I'll go with the league average, and I'll add a standard deviation since they probably we're exactly middle-of-the-road. Probably not the best way but it's only a few datapoints.

[233]:
```
baopp_checks = df[df['retroID'].isin(df_baopp_missing['retroID'])].
↪sort_values('retroID')
```

[234]:
```
baopp_checks['retroID'].value_counts()
```

[234]:
```
radis001    11
arrof001     9
brotr001     7
tolls002     5
apodb101     5
villb002     4
dunnj001     2
eschj001     2
halts001     2
younl101     1
glasn101     1
fordw103     1
Name: retroID, dtype: int64
```

We only have three data points with one year of appearances. We'll fill those with the league average.

[235]:
```
val_counts = baopp_checks['retroID'].value_counts()
```

[236]:
```
from itertools import compress
# Get list of retroIDs of players who only have one year appearance
```

[237]:
```
one_time_players = list(compress(val_counts.index, val_counts.eq(1)))
```

[238]:
```
df[df['retroID'].isin(one_time_players)]
```

[238]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
297	glasn101	1920	0	0	1	0	0	0	0	7	...	0.0	0	1	
3709	fordw103	1936	0	0	1	0	0	0	0	0	...	0.0	0	0	
13621	younl101	1971	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
297	0	12.0	0	4	0.0	0.0	0.0
3709	0	3.0	0	2	0.0	0.0	0.0
13621	0	0.0	0	0	0.0	0.0	0.0

[3 rows x 27 columns]

[239]:
```
df['BAOpp'].mean()
```

[239]:
```
0.27445659068384537
```

[240]:
```
df['BAOpp'].std()
```

[240]:
```
0.07751058079199835
```

[241]:
```
filled_baopp = df['BAOpp'].mean() + df['BAOpp'].std()
```

[242]:
```
filled_baopp
```

[242]:
```
0.3519671714758437
```

[243]:
```
df.loc[df['retroID'].isin(one_time_players), ['BAOpp']] = filled_baopp
```

[244]:
```
df[df['retroID'].isin(one_time_players)]['BAOpp']
```

[244]:
```
297      0.351967
3709     0.351967
13621    0.351967
Name: BAOpp, dtype: float64
```

[245]:
```
df_baopp_missing = df[df['BAOpp'].isnull()].sort_values('retroID')
```

[246]:
```
df_baopp_missing
```

[246]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
14000	apodb101	1973	0	0	1	0	0	0	0	0	...	0.0	0	0	
19114	arrof001	1986	0	0	1	0	0	0	0	0	...	0.0	0	0	
39581	brotr001	2018	0	0	1	0	0	0	0	0	...	0.0	0	0	
36447	dunnj001	2014	0	0	1	0	0	0	0	2	...	0.0	2	0	
38848	eschj001	2017	0	0	1	0	0	0	0	0	...	0.0	0	0	
26791	halts001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
27036	radis001	2000	0	0	1	0	0	0	0	0	...	0.0	0	0	
36189	tolls002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	
36208	villb002	2013	0	0	1	0	0	0	0	0	...	0.0	0	0	

	BK	BFP	GF	R	SH	SF	GIDP
14000	0	2.0	0	1	0.0	0.0	0.0
19114	0	3.0	0	0	0.0	0.0	0.0
39581	0	2.0	0	1	0.0	0.0	0.0
36447	0	2.0	0	0	0.0	1.0	0.0
38848	0	2.0	0	0	0.0	0.0	0.0
26791	0	1.0	0	0	0.0	0.0	0.0
27036	0	1.0	0	0	0.0	0.0	0.0
36189	0	2.0	0	0	0.0	0.0	0.0
36208	0	1.0	1	0	0.0	0.0	0.0

[9 rows x 27 columns]

Now we just have to worry about the players with at least two years of appearances.

[247]:
```
baopp_checks = df[df['retroID'].isin(df_baopp_missing['retroID'])].
↪sort_values('retroID')
```

[248]:
```
baopp_checks['retroID'].value_counts()
```

[248]:
```
radis001    11
arrof001     9
brotr001     7
tolls002     5
apodb101     5
villb002     4
dunnj001     2
eschj001     2
halts001     2
Name: retroID, dtype: int64
```

[249]:
```
one_time_players = list(val_counts.index)
```

[250]:
```
one_time_players
```

[250]:
```
['radis001',
 'arrof001',
 'brotr001',
 'tolls002',
 'apodb101',
 'villb002',
 'dunnj001',
 'eschj001',
 'halts001',
 'younl101',
 'glasn101',
 'fordw103']
```

[251]:
```
df[df['retroID'] == 'radis001']['BAOpp']
```

[251]:
```
21355    0.241
21865    0.206
22335    0.243
22871    0.268
23957    0.309
24557    0.264
25145    0.236
25740    0.272
26377    0.270
27036      NaN
27708    0.400
Name: BAOpp, dtype: float64
```

[252]:
```
df[df['retroID'] == 'radis001']['BAOpp'].mean().round(4)
```

[252]:
```
0.2709
```

This looks good, so we'll iterate through and assign each player's missing BAOpp as his career

mean for that state.

[253]:
```
df['BAOpp'] = df.groupby("retroID")['BAOpp'].transform(lambda baopp: baopp.
↪fillna(baopp.mean()))
```

[254]:
```
100 * df.isnull().sum() / len(df)
```

[254]:
```
retroID    0.000000
yearID     0.000000
W          0.000000
L          0.000000
G          0.000000
GS         0.000000
CG         0.000000
SHO        0.000000
SV         0.000000
IPouts     0.000000
H          0.000000
ER         0.000000
HR         0.000000
BB         0.000000
SO         0.000000
BAOpp      0.000000
ERA        0.183295
IBB        0.000000
WP         0.000000
HBP        0.000000
BK         0.000000
BFP        0.007431
GF         0.000000
R          0.000000
SH         0.000000
SF         0.000000
GIDP       0.000000
dtype: float64
```

[255]:
```
df.head()
```

[255]:

	retroID	yearID	W	L	G	GS	CG	SHO	SV	IPouts	...	IBB	WP	HBP	\
0	adamb104	1919	17	10	34	29	23	6	1	790	...	0.0	2	3	
1	adamw101	1919	0	0	1	0	0	0	0	14	...	0.0	0	1	
2	alexg102	1919	16	11	30	27	20	9	1	705	...	0.0	1	0	
3	altrn101	1919	0	0	1	0	0	0	0	0	...	0.0	0	0	
4	amesr101	1919	3	5	23	7	1	0	1	210	...	0.0	3	1	

	BK	BFP	GF	R	SH	SF	GIDP
0	0	1017.0	5	66	0.0	0.0	0.0
1	0	21.0	1	2	0.0	0.0	0.0

```
2   0   906.0   3  51  0.0  0.0   0.0
3   0     4.0   0   4  0.0  0.0   0.0
4   0   314.0  10  44  0.0  0.0   0.0

[5 rows x 27 columns]
```

Missing Values: ERA

[256]:
```
df_era_missing = df[df['ERA'].isnull()].sort_values('retroID')
```

[257]:
```
df_era_missing
```

[257]:
```
        retroID  yearID  W  L  G  GS  CG  SHO  SV  IPouts  ...  IBB  WP  HBP  \
3      altrn101    1919  0  0  1   0   0    0   0       0  ...  0.0   0    0
20491  alvaw001    1989  0  1  1   1   0    0   0       0  ...  0.0   0    0
14000  apodb101    1973  0  0  1   0   0    0   0       0  ...  0.0   0    0
19114  arrof001    1986  0  0  1   0   0    0   0       0  ...  0.0   0    0
663    bents101    1922  0  0  1   0   0    0   0       0  ...  0.0   0    0
...         ...     ... .. .. ..  ..  ..  ...  ..     ...  ...  ...  ..  ...
40336  weisz001    2018  0  0  1   0   0    0   0       0  ...  0.0   0    0
6277   willa103    1946  0  0  1   0   0    0   0       0  ...  0.0   0    0
10476  willt102    1962  0  0  1   0   0    0   0       0  ...  0.0   0    0
18235  wortr101    1983  0  0  1   0   0    0   0       0  ...  0.0   0    0
13621  younl101    1971  0  0  1   0   0    0   0       0  ...  0.0   0    0

       BK  BFP  GF  R   SH   SF  GIDP
3       0  4.0   0  4  0.0  0.0   0.0
20491   0  5.0   0  3  0.0  0.0   0.0
14000   0  2.0   0  1  0.0  0.0   0.0
19114   0  3.0   0  0  0.0  0.0   0.0
663     0  2.0   0  0  0.0  0.0   0.0
...    ..  ...  .. ..  ...  ...   ...
40336   0  4.0   0  4  0.0  0.0   0.0
6277    0  2.0   0  0  0.0  0.0   0.0
10476   0  3.0   0  1  0.0  0.0   0.0
18235   0  4.0   0  1  0.0  0.0   0.0
13621   0  0.0   0  0  0.0  0.0   0.0

[74 rows x 27 columns]
```

[258]:
```
df_era_missing['retroID'].nunique()
```

[258]:
```
74
```

74 rows and 74 unique IDs means that each of these players is only missing the ERA stat for one year. We'll first see, like with BAOpp, if they played other years.

[259]:
```
era_checks = df[df['retroID'].isin(df_era_missing['retroID'])].
 ↪sort_values('retroID')
```

[260]:
```
val_counts = era_checks['retroID'].value_counts()
```

[261]:
```
one_time_players = list(compress(val_counts.index, val_counts.eq(1)))
```

[262]:
```
one_time_players
```

[262]:
```
['russr102',
 'moorb104',
 'palam101',
 'musis101',
 'garda103',
 'weisz001',
 'koenw101',
 'bents101',
 'fordw103',
 'schej101',
 'brucf101',
 'davav101',
 'hamad101',
 'walkm101',
 'sundg101',
 'younl101',
 'browj102']
```

[263]:
```
df['ERA'].mean()
```

[263]:
```
5.165393567919002
```

[264]:
```
df['ERA'].std()
```

[264]:
```
5.2791159271962815
```

Intuitively, 5.17 is a bit of a high ERA. Though the stat can grow infinitely in theory and low numbers are very difficult, I don't want to assign 10 to the missing values. It's just too much. I'll just do mean + std/2.

[265]:
```
filled_era = df['ERA'].mean() + (df['ERA'].std())/2
```

[266]:
```
df.loc[df['retroID'].isin(one_time_players), ['ERA']] = filled_era
df[df['retroID'].isin(one_time_players)]['ERA']
```

[266]:
```
101      7.804952
173      7.804952
663      7.804952
```

```
722      7.804952
931      7.804952
1447     7.804952
1755     7.804952
2154     7.804952
3709     7.804952
4485     7.804952
4513     7.804952
7678     7.804952
8773     7.804952
9903     7.804952
12532    7.804952
13621    7.804952
40336    7.804952
Name: ERA, dtype: float64
```

[267]:
```
df_era_missing = df[df['ERA'].isnull()].sort_values('retroID')
```

We'll continue to follow the same method as for BAOpp with the rest of the missing values.

[268]:
```
df_era_missing.shape
```

[268]:
```
(57, 27)
```

[269]:
```
era_checks = df[df['retroID'].isin(df_era_missing['retroID'])].
 ↪sort_values('retroID')
era_checks['retroID'].value_counts()
```

[269]:
```
hillr001    16
choar001    16
mclic101    15
alvaw001    15
farme101    14
medid101    14
kosld101    13
coopm101    13
radis001    11
burkb102    10
owchb001    10
milna101    10
arrof001     9
chent101     9
harvb001     9
perim001     9
deanp101     9
ray-j101     9
pennb001     7
```

```
navaj101     7
brotr001     7
jonen001     7
painp101     6
willt102     6
moorc101     6
tolls002     5
luebs101     5
reina102     5
scarm101     5
mccud001     5
apodb101     5
blake101     4
villb002     4
kreur101     4
wortr101     4
tankd001     3
pitls101     3
stufp101     3
sabee001     3
geard101     3
kochm001     3
vaugp101     3
roeto101     2
urdal001     2
willa103     2
green002     2
dibup101     2
uhl-b101     2
wardd101     2
kella101     2
eschj001     2
kammb101     2
jeant101     2
engej101     2
smitd105     2
halts001     2
altrn101     2
Name: retroID, dtype: int64
```

[270]:
```
df['ERA'] = df.groupby("retroID")['ERA'].transform(lambda era: era.fillna(era.
 ↪mean()))
```

[271]:
```
100 * df.isnull().sum() / len(df)
```

[271]:
```
retroID     0.000000
yearID      0.000000
```

```
W           0.000000
L           0.000000
G           0.000000
GS          0.000000
CG          0.000000
SHO         0.000000
SV          0.000000
IPouts      0.000000
H           0.000000
ER          0.000000
HR          0.000000
BB          0.000000
SO          0.000000
BAOpp       0.000000
ERA         0.000000
IBB         0.000000
WP          0.000000
HBP         0.000000
BK          0.000000
BFP         0.007431
GF          0.000000
R           0.000000
SH          0.000000
SF          0.000000
GIDP        0.000000
dtype: float64
```

Missing Values: BFP

[272]:
```
df_bfp_missing = df[df['BFP'].isnull()].sort_values('retroID')
```

[273]:
```
df_bfp_missing
```

[273]:
```
       retroID  yearID  W  L   G  GS  CG  SHO  SV  IPouts  ...  IBB  WP  HBP  \
709   fourj101    1922  0  0   1   0   0    0   0       3  ...  0.0   0    0
1171  jamel101    1924  0  0   1   0   0    0   0       3  ...  0.0   0    0
802   pierb103    1922  3  9  29  12   7    1   0     364  ...  0.0   4    6

      BK  BFP  GF   R   SH   SF  GIDP
709    0  NaN   1   0  0.0  0.0   0.0
1171   0  NaN   1   2  0.0  0.0   0.0
802    0  NaN  10  77  0.0  0.0   0.0

[3 rows x 27 columns]
```

These amounts are negligable. Rather than take averages or set to 0, I'm going to get a little clever. A pitcher intuitively faces hitters until he gets an out, and comes out of the game if he can't get one. There's a lot of work I could do to get a good approximation, but since I'm only filling 3 rows

out of over 40,000 I'm just going to set the missing values to (IPouts - G). This gives us the number of outs a pitcher earned minus 1 for each game he appeared in (presumably this 1 represents the final batter, whom the pitcher did not get out).

[274]:
```
df['BFP'].fillna(df['IPouts'] - df['G'], inplace=True)
```

[275]:
```
100 * df.isnull().sum() / len(df)
```

[275]:
```
retroID    0.0
yearID     0.0
W          0.0
L          0.0
G          0.0
GS         0.0
CG         0.0
SHO        0.0
SV         0.0
IPouts     0.0
H          0.0
ER         0.0
HR         0.0
BB         0.0
SO         0.0
BAOpp      0.0
ERA        0.0
IBB        0.0
WP         0.0
HBP        0.0
BK         0.0
BFP        0.0
GF         0.0
R          0.0
SH         0.0
SF         0.0
GIDP       0.0
dtype: float64
```

Data Aggregation

Now we can group by retroID, but we need to be more careful than we were with fielding, catching and batting. Some of these stats are averages and some are sum totals, so when we group by we need to handle them differently. We'll split them into two dataframes and do a join. The splitting step will require some intuitive knowledge about baseball statistics. But before we do all of this, we can now get rid of the yearID column.

[276]:
```
df.drop(columns=['yearID'], inplace=True)
```

[277]:
```
df.columns
```

[277]:
```
Index(['retroID', 'W', 'L', 'G', 'GS', 'CG', 'SHO', 'SV', 'IPouts', 'H', 'ER',
       'HR', 'BB', 'SO', 'BAOpp', 'ERA', 'IBB', 'WP', 'HBP', 'BK', 'BFP', 'GF',
       'R', 'SH', 'SF', 'GIDP'],
      dtype='object')
```

[278]:
```
average_stats = ['BAOpp', 'ERA']
```

It's only two columns that are averages, and we could probably do without ERA since it's a function of batters faced and runs allowed. We'll keep it since it's such a fundamental statistic in the sport and we have to split anyway for BAOpp, which is a very important one to keep track of.

[279]:
```
df_avgs = df[['retroID', 'BAOpp', 'ERA']]
```

[280]:
```
df_avgs.head()
```

[280]:

	retroID	BAOpp	ERA
0	adamb104	0.22	1.98
1	adamw101	0.38	3.86
2	alexg102	0.21	1.72
3	altrn101	1.00	0.00
4	amesr101	0.31	4.89

[281]:
```
df_sums = df.drop(columns=average_stats)
```

[282]:
```
df_sums.head()
```

[282]:

	retroID	W	L	G	GS	CG	SHO	SV	IPouts	H	...	IBB	WP	HBP	BK	\
0	adamb104	17	10	34	29	23	6	1	790	213	...	0.0	2	3	0	
1	adamw101	0	0	1	0	0	0	0	14	7	...	0.0	0	1	0	
2	alexg102	16	11	30	27	20	9	1	705	180	...	0.0	1	0	0	
3	altrn101	0	0	1	0	0	0	0	0	4	...	0.0	0	0	0	
4	amesr101	3	5	23	7	1	0	1	210	88	...	0.0	3	1	0	

	BFP	GF	R	SH	SF	GIDP
0	1017.0	5	66	0.0	0.0	0.0
1	21.0	1	2	0.0	0.0	0.0
2	906.0	3	51	0.0	0.0	0.0
3	4.0	0	4	0.0	0.0	0.0
4	314.0	10	44	0.0	0.0	0.0

[5 rows x 24 columns]

[283]:
```
df_avgs.shape
```

[283]:
```
(40372, 3)
```

[284]:
```
df_sums.shape
```

[284]:
```
(40372, 24)
```

[289]:
```
df_avgs = df_avgs.groupby('retroID').mean().round(4).reset_index()
```

[292]:
```
df_avgs.shape
```

[292]:
```
(7835, 3)
```

[288]:
```
df_sums = df_sums.groupby('retroID').sum().reset_index()
```

[293]:
```
df_sums.shape
```

[293]:
```
(7835, 24)
```

[291]:
```
pd.merge(df_avgs, df_sums, on='retroID')
```

[291]:

	retroID	BAOpp	ERA	W	L	G	GS	CG	SHO	SV	...	IBB	WP	\
0	aardd001	0.2574	5.1944	16	18	331	0	0	0	69	...	22.0	12	
1	aased001	0.2508	3.4931	66	60	448	91	22	5	82	...	45.0	22	
2	abadf001	0.2501	4.0733	8	27	363	6	0	0	2	...	10.0	9	
3	abbog001	0.2786	4.3317	62	83	248	206	37	5	0	...	28.0	18	
4	abboj001	0.2804	4.4964	87	108	263	254	31	6	0	...	30.0	53	
...	...	...	...	..	...	...	...	..	...	..	...	...	..	
7830	zolds101	0.2700	3.6890	43	53	250	93	30	5	8	...	0.0	8	
7831	zubeb101	0.2717	5.3617	43	42	224	65	23	3	6	...	0.0	28	
7832	zumaj001	0.2286	3.4420	13	12	171	0	0	0	5	...	11.0	16	
7833	zuveg101	0.2760	4.1280	32	36	265	31	9	2	40	...	29.0	10	
7834	zycht001	0.2183	2.8000	7	3	70	1	0	0	1	...	5.0	2	

	HBP	BK	BFP	GF	R	SH	SF	GIDP
0	16	1	1475.0	141	169	17.0	11.0	21.0
1	7	3	4730.0	235	503	50.0	34.0	106.0
2	12	2	1350.0	96	137	7.0	12.0	22.0
3	32	5	5508.0	13	707	60.0	39.0	111.0
4	32	11	7211.0	5	880	70.0	47.0	200.0
...	...	..	...	...	...	...	...	...
7830	3	4	3946.0	78	423	0.0	0.0	0.0
7831	4	1	3476.0	90	418	0.0	0.0	0.0
7832	4	0	911.0	35	80	6.0	10.0	10.0
7833	27	1	2746.0	139	296	0.0	0.0	0.0
7834	8	1	309.0	14	24	1.0	3.0	6.0

[7835 rows x 26 columns]

This gives us the appropriate amount of rows and columns, so the merge worked. We'll send this as our final output.

[294]:
```
df = pd.merge(df_avgs, df_sums, on='retroID')
```

[296]:
```
df.shape
```

[296]:
```
(7835, 26)
```

We're ready to export the resulting table by saving to a csv.