Predicting Wins in MLB

Author: Michael Tiernan

Brief History

Betting on sports has a long history despite the fact that it has only recently become legal to do so outside of Nevada (which legalized it in 1942). New Jersey became just the second state in the US to legalize sports betting following a US Supreme Court ruling in May of 2018. This was a major court decision that opened the floodgates for more states to follow and billions of dollars to be generated doing so. As of today, June 24, 2021, 22 states have legalized sports betting in the US. The rest of the US has either passed or introduced a bill to legalize the industry with the exception of only three states (Idaho, Utah, Wisconsin) who have not.

For those not familiar with the industry, a sportsbook (ex. DraftKings, William Hill, FanDuel, Barstool) creates betting odds based on their own machine learning models for the user to then wager money on. If the two teams are very similar and there is no clear favorite, the odds for that game would be close to even. When the odds are even, a $100betwillreturna$100 payout (if the team you bet on wins). But what if one team is heavily favored? When the best team has a game against the worst team, the odds for that game have to account for this disparity and the way they do that is adjusting the odds to lets say -200, which means that your $100betnowonlyreturnsa$50 payout. By doing so, one may be persuaded to then bet on the underdog in this scenario because the odds on the underdog will be +150, which means your $100wouldreturn$150 payout. By adjusting the odds, the sportsbook is looking to have an even amount of action on each team because this almost assures them a profit due to the fact that it is expected that the favorite will win, in which case they only have to payout $50inthe2ndscenario, butatthesametimecollectingthe$100 lost bet from the underdog bettor. This, coupled with the resources and mastery that the sportsbooks have in creating the odds, is what makes betting on sports so difficult in terms of being profitable over the long-term for the average user.

Business Problem

Given this brief history, it is clear that the average bettor has an uphill battle to long-term profitability. For my analysis, I am looking to answer the two questions below:

```
1) What are the best predictors for season total wins?
2) What are the best predictors for individual game wins?
```

Like any other game, whether it be casino, board games, or competitive sport, the intelligent player is going to look for a competitive advantage. Within the sports betting industry, the inexperienced bettor likely makes their wagers based off of gut intuition. The intermediate bettor may incorporate research from articles read or a comparison of some surface-level stats, but the true competitive advantage comes from a deep-dive into the data, using machine learning algorithms to account for as many factors/predictors as possible.

This is the strategy I will continuously build upon in pursuit of creating a highly competitive algorithm for long-term sports betting profitability.

In [1]:
```
# importing pacakges for analysis
import pandas as pd
# setting pandas display to avoid scientific notation
pd.options.display.float_format = '{:.3f}'.format
# setting to see all columns in outputs of dataframes
pd.set_option('display.max_columns', None)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_theme(style="whitegrid")
import warnings
warnings.filterwarnings('ignore')
```

Yearly Stats

This first dataset comes from the lahman's baseball database. This is the go-to database for baseball analysis. This particular dataset that I am going to work with has yearly data for each team from 1871 - 2020. There are 48 columns of stats and information to get me started.

The Covid-19 pandemic caused the 2020 season to be shortened to just 60 games (a regular full season is 162 games) and some players opting out of playing. Given this circumstance, I chose to remove this shortened year from my analysis.

In [2]:
```
df_teams = pd.read_csv('data/Teams.csv')
df_teams.tail()
```

Out[2]:

	yearID	lgID	teamID	franchID	divID	Rank	G	Ghome	W	L	DivWin	WCWin	LgWin
2950	2020	NL	SLN	STL	C	3	58	27.000	30	28	N	Y	N
2951	2020	AL	TBA	TBD	E	1	60	29.000	40	20	Y	N	Y
2952	2020	AL	TEX	TEX	W	5	60	30.000	22	38	N	N	N
2953	2020	AL	TOR	TOR	E	3	60	26.000	32	28	N	Y	N
2954	2020	NL	WAS	WSN	E	4	60	33.000	26	34	N	N	N

In [3]:
```
# drop cols
cols = ['teamID','lgID','divID','Rank','G','Ghome','L','CS','HBP','IPouts','name'
df_teams.drop(cols, axis=1, inplace=True)
df_teams.tail()
```

Out[3]:

	yearID	franchID	W	DivWin	WCWin	LgWin	WSWin	R	AB	H	2B	3B	HR	B
2950	2020	STL	30	N	Y	N	N	240	1752	410	73	7	51	205.0(
2951	2020	TBD	40	Y	N	Y	N	289	1975	470	105	12	80	243.0(
2952	2020	TEX	22	N	N	N	N	224	1936	420	80	9	62	167.0(
2953	2020	TOR	32	N	Y	N	N	302	2023	516	104	4	88	203.0(
2954	2020	WSN	26	N	N	N	N	293	1968	519	112	12	66	192.0(

In [4]:
```
# dummy championships
cols = ['DivWin','WCWin','LgWin','WSWin']
dummy_df = pd.get_dummies(df_teams[cols], drop_first=True)
dummy_df.tail()
```

Out[4]:

	DivWin_Y	WCWin_Y	LgWin_Y	WSWin_Y
2950	0	1	0	0
2951	1	0	1	0
2952	0	0	0	0
2953	0	1	0	0
2954	0	0	0	0

In [5]:
```
df_teams = pd.concat([df_teams, dummy_df], axis=1)
df_teams.drop(['DivWin','WCWin','LgWin','WSWin'], axis=1, inplace=True)
df_teams.tail()
```

Out[5]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	SO	SB	SF	RA
2950	2020	STL	30	240	1752	410	73	7	51	205.000	477.000	18.000	16.000	229
2951	2020	TBD	40	289	1975	470	105	12	80	243.000	608.000	48.000	14.000	229
2952	2020	TEX	22	224	1936	420	80	9	62	167.000	548.000	49.000	18.000	312
2953	2020	TOR	32	302	2023	516	104	4	88	203.000	508.000	33.000	14.000	312
2954	2020	WSN	26	293	1968	519	112	12	66	192.000	451.000	33.000	21.000	301

In [6]:
```
# feature engineer team run diff and batting avg.
df_teams['run_diff'] = (df_teams['R'] - df_teams['RA'])
df_teams['ba'] = (df_teams['H'] / df_teams['AB'])
df_teams.tail()
```

Out[6]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	SO	SB	SF	RA
2950	2020	STL	30	240	1752	410	73	7	51	205.000	477.000	18.000	16.000	229
2951	2020	TBD	40	289	1975	470	105	12	80	243.000	608.000	48.000	14.000	229
2952	2020	TEX	22	224	1936	420	80	9	62	167.000	548.000	49.000	18.000	312
2953	2020	TOR	32	302	2023	516	104	4	88	203.000	508.000	33.000	14.000	312
2954	2020	WSN	26	293	1968	519	112	12	66	192.000	451.000	33.000	21.000	301

In [7]:
```
# filter db to years 2000 - 2019
df_teams = df_teams.loc[(df_teams['yearID']>1999) & (df_teams['yearID']<2020)]
df_teams
```

Out[7]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	SO	SB	SF
2325	2000	ANA	82	864	5628	1574	309	34	236	608.000	1024.000	93.000	43.000
2326	2000	ARI	85	792	5527	1466	282	44	179	535.000	975.000	97.000	58.000
2327	2000	ATL	95	810	5489	1490	274	26	179	595.000	1010.000	148.000	45.000
2328	2000	BAL	74	794	5549	1508	310	22	184	558.000	900.000	126.000	54.000
2329	2000	BOS	85	792	5630	1503	316	32	167	611.000	1019.000	43.000	48.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2920	2019	STL	91	764	5449	1336	246	24	210	561.000	1420.000	116.000	39.000
2921	2019	TBD	96	769	5628	1427	291	29	217	542.000	1493.000	94.000	34.000
2922	2019	TEX	78	810	5540	1374	296	24	223	534.000	1578.000	131.000	44.000
2923	2019	TOR	67	726	5493	1299	270	21	247	509.000	1514.000	51.000	28.000
2924	2019	WSN	93	873	5512	1460	298	27	231	584.000	1308.000	116.000	42.000

In [8]:
```
nym = df_teams.loc[df_teams['franchID']=='NYM']
nym
```

Out[8]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	SO	SB	SF	R
2344	2000	NYM	94	807	5486	1445	281	20	198	675.000	1037.000	66.000	51.000	7
2374	2001	NYM	82	642	5459	1361	273	18	147	545.000	1062.000	66.000	35.000	7
2404	2002	NYM	75	690	5496	1409	238	22	160	486.000	1044.000	87.000	30.000	7
2434	2003	NYM	66	642	5341	1317	262	24	124	489.000	1035.000	70.000	45.000	7
2464	2004	NYM	71	684	5532	1376	289	20	185	512.000	1159.000	107.000	34.000	7
2493	2005	NYM	83	722	5505	1421	279	32	175	486.000	1075.000	153.000	38.000	6
2523	2006	NYM	97	834	5558	1469	323	41	200	547.000	1071.000	146.000	47.000	7
2553	2007	NYM	88	804	5605	1543	294	27	177	549.000	981.000	200.000	58.000	7
2583	2008	NYM	89	799	5606	1491	274	38	172	619.000	1024.000	138.000	49.000	7
2613	2009	NYM	70	671	5453	1472	295	49	95	526.000	928.000	122.000	55.000	7
2643	2010	NYM	79	656	5465	1361	266	40	128	502.000	1095.000	130.000	57.000	6
2673	2011	NYM	77	718	5600	1477	309	39	108	571.000	1085.000	130.000	48.000	7
2703	2012	NYM	74	650	5450	1357	286	21	139	503.000	1250.000	79.000	30.000	7
2733	2013	NYM	74	619	5559	1318	263	32	130	512.000	1384.000	114.000	32.000	6
2763	2014	NYM	79	629	5472	1306	275	19	125	516.000	1264.000	101.000	44.000	6
2793	2015	NYM	90	683	5527	1351	295	17	177	488.000	1290.000	51.000	32.000	6
2823	2016	NYM	87	671	5459	1342	240	19	218	517.000	1302.000	42.000	41.000	6
2853	2017	NYM	70	735	5510	1379	286	28	224	529.000	1291.000	58.000	37.000	8
2883	2018	NYM	77	676	5468	1282	265	34	170	566.000	1404.000	71.000	42.000	7
2913	2019	NYM	86	791	5624	1445	280	17	242	516.000	1384.000	56.000	27.000	7

Which team has averaged the most amount of wins over the past 20 years?

As a Mets fan, I begrudgingly hypothesize that the Yankees is our answer.

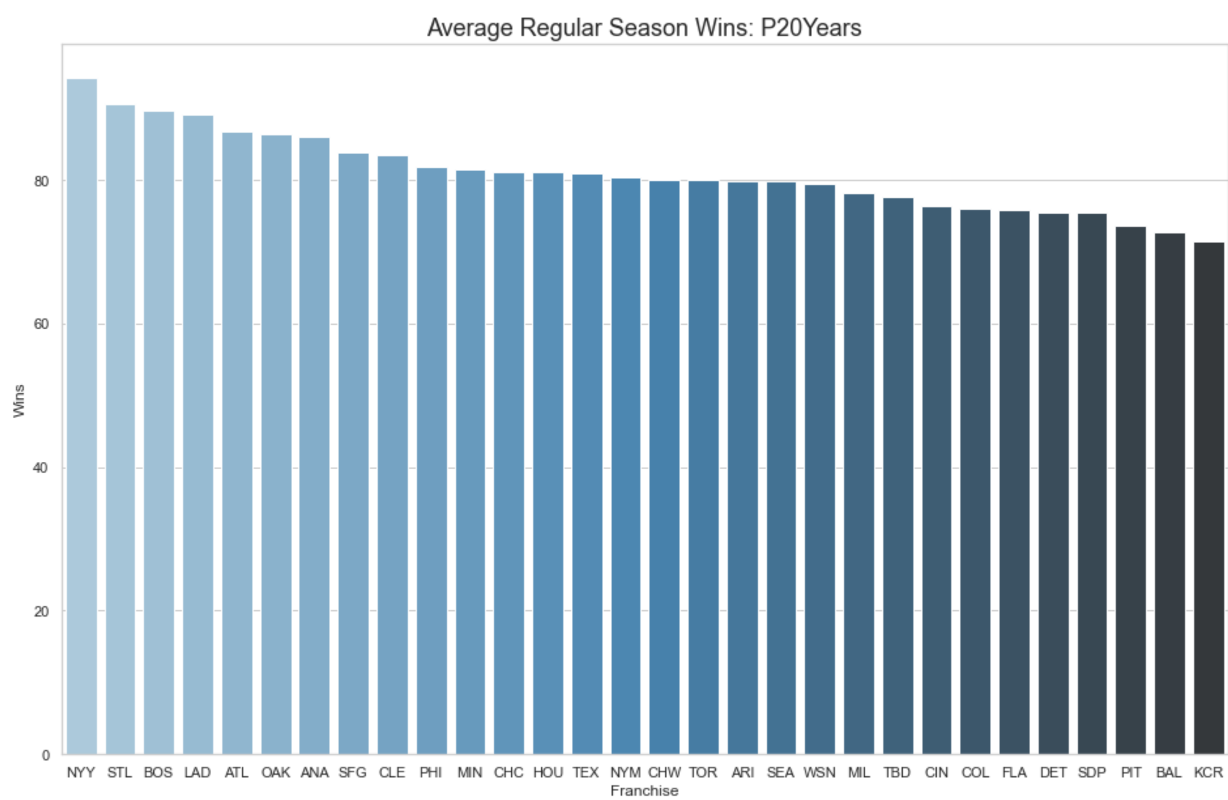
In [9]:
```
wins_df = df_teams.groupby(['franchID'])['W'].mean()
```

In [10]:
```
wins_df = pd.DataFrame(wins_df).sort_values('W', ascending=False)
wins_df
```

Out[10]:

	W
franchID	
NYY	94.300
STL	90.600
BOS	89.600
LAD	89.050
ATL	86.750
OAK	86.450
ANA	86.100
SFG	83.800
CLE	83.550
PHI	81.850

In [13]:
```
fig,ax = plt.subplots(figsize=(16,10))
ax = sns.barplot(x=wins_df.index, y='W', data=wins_df, palette='Blues_d')
ax.set_title("Average Regular Season Wins: P20Years", fontsize=18)
ax.set(xlabel='Franchise', ylabel='Wins')
plt.show()
```



What about past 5 years?

In [15]:
```
# filter db to P5Years, 2015 - 2019
df_p5y = df_teams.loc[(df_teams['yearID']>2014)]
df_p5y
```

Out[15]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	SO	SB	SF	R
2775	2015	ARI	79	720	5649	1494	289	48	154	490.000	1312.000	132.000	57.000	7
2776	2015	ATL	67	573	5420	1361	251	18	100	471.000	1107.000	69.000	31.000	76
2777	2015	BAL	81	713	5485	1370	246	20	217	418.000	1331.000	44.000	32.000	69
2778	2015	BOS	78	748	5640	1495	294	33	161	478.000	1148.000	71.000	42.000	75
2779	2015	CHW	76	622	5533	1381	260	27	136	404.000	1231.000	68.000	37.000	70
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2920	2019	STL	91	764	5449	1336	246	24	210	561.000	1420.000	116.000	39.000	66
2921	2019	TBD	96	769	5628	1427	291	29	217	542.000	1493.000	94.000	34.000	65
2922	2019	TEX	78	810	5540	1374	296	24	223	534.000	1578.000	131.000	44.000	87
2923	2019	TOR	67	726	5493	1299	270	21	247	509.000	1514.000	51.000	28.000	82
2924	2019	WSN	93	873	5512	1460	298	27	231	584.000	1308.000	116.000	42.000	72

150 rows × 34 columns

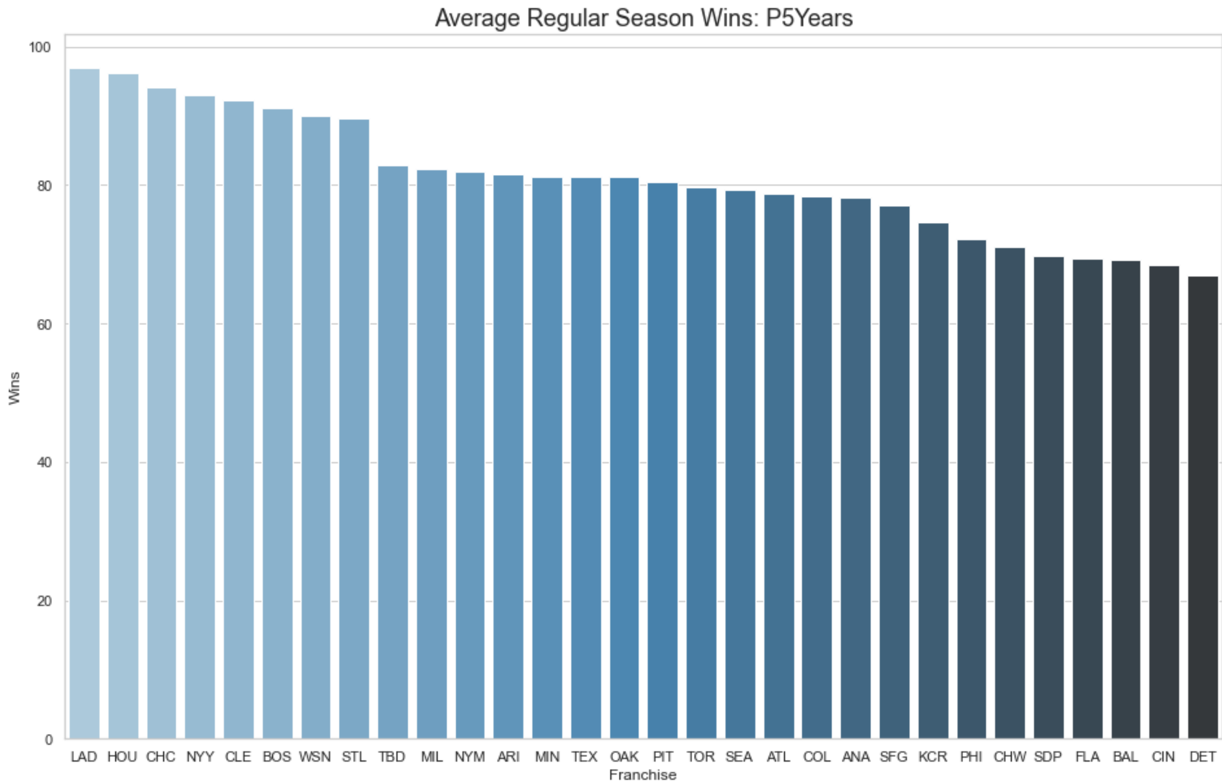
In [17]:
```
df_p5y_grouped = df_p5y.groupby(['franchID'])['W'].mean()
```

In [18]:
```
df_p5y_grouped = pd.DataFrame(df_p5y_grouped).sort_values('W', ascending=False)
df_p5y_grouped
```

Out[18]:

franchID	W
LAD	97.000
HOU	96.200
CHC	94.200
NYY	93.000
CLE	92.200
BOS	91.200
WSN	90.000
STL	89.600
TBD	82.800
MIL	82.400

In [19]:
```
# P5Years
fig,ax = plt.subplots(figsize=(16,10))
ax = sns.barplot(x=df_p5y_grouped.index, y='W', data=df_p5y_grouped, palette='Blu
ax.set_title("Average Regular Season Wins: P5Years", fontsize=18)
ax.set(xlabel='Franchise', ylabel='Wins')
plt.show()
```



In [20]:
```
# P20Years and P5Years dfs
print(df_teams.shape)
print(df_p5y.shape)
```

```
(600, 34)
(150, 34)
```

In [21]:
```
df_p5y
```

Out[21]:

	yearID	franchID	W	R	AB	H	2B	3B	HR	BB	...	DP	FP	BPF	PPF	Div'
2775	2015	ARI	79	720	5649	1494	289	48	154	490.0	...	146	0.986	107	106	
2776	2015	ATL	67	573	5420	1361	251	18	100	471.0	...	186	0.985	97	97	
2777	2015	BAL	81	713	5485	1370	246	20	217	418.0	...	134	0.987	103	104	
2778	2015	BOS	78	748	5640	1495	294	33	161	478.0	...	148	0.984	104	107	
2779	2015	CHW	76	622	5533	1381	260	27	136	404.0	...	159	0.983	92	93	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2920	2019	STL	91	764	5449	1336	246	24	210	561.0	...	168	0.989	98	97	
2921	2019	TBD	96	769	5628	1427	291	29	217	542.0	...	126	0.985	97	96	
2922	2019	TEX	78	810	5540	1374	296	24	223	534.0	...	143	0.982	111	112	
2923	2019	TOR	67	726	5493	1299	270	21	247	509.0	...	141	0.984	97	98	
2924	2019	WSN	93	873	5512	1460	298	27	231	584.0	...	111	0.985	106	104	

150 rows × 34 columns

Advanced Stats

Baseballsavant.com allows you to dynamically pull stats. For my analysis I'm going to use linedrivepercentage, xOBP, and xwOBA.

```
- linedrivepercentage = percentage of time a team hit a linedrive. Linedr
ives are a hard-hit ball that has the best opportunity of the batter to s
uccessfully reach base, as opposed to groundballs and flyballs.

- xOBP = expected on base percentage, which is formulated by quality of c
ontact, strikeouts, and walks

- xwOBA = expected weighted on base average, which is formulated using ex
it velocity, launch angle, and on certain types of batted balls, sprint s
peed
```

In [21]:
```
df_newstats = pd.read_csv('data/new_stats.csv')
df_newstats
```

Out[21]:

	franchID	yearID	linedrivepercent	xobp	xwoba
0	ANA	2015	27.200	0.304	0.304
1	ARI	2015	29.100	0.306	0.303
2	ATL	2015	25.800	0.307	0.294
3	BAL	2015	26.400	0.297	0.307
4	BOS	2015	23.900	0.307	0.304
...	...	...	...	...	...
145	STL	2019	24.900	0.317	0.315
146	TBD	2019	25.600	0.326	0.327
147	TEX	2019	25.600	0.315	0.317
148	TOR	2019	24.500	0.307	0.312
149	WSN	2019	25.100	0.334	0.334

150 rows × 5 columns

In [46]:
```
xwoba = df_newstats.groupby(['franchID','yearID'])['xwoba'].mean()
```

In [66]:
```
xwoba_df = pd.DataFrame(xwoba).sort_values(['yearID','xwoba'], ascending=False)
xwoba_df.head(10)
```

Out[66]:

		xwoba
franchID	**yearID**	
MIN	**2019**	0.345
ATL	**2019**	0.336
WSN	**2019**	0.334
LAD	**2019**	0.333
NYY	**2019**	0.333
HOU	**2019**	0.332
BOS	**2019**	0.331
OAK	**2019**	0.331
TBD	**2019**	0.327
CHC	**2019**	0.324

The Minnesota Twins record in 2019 was 101-61

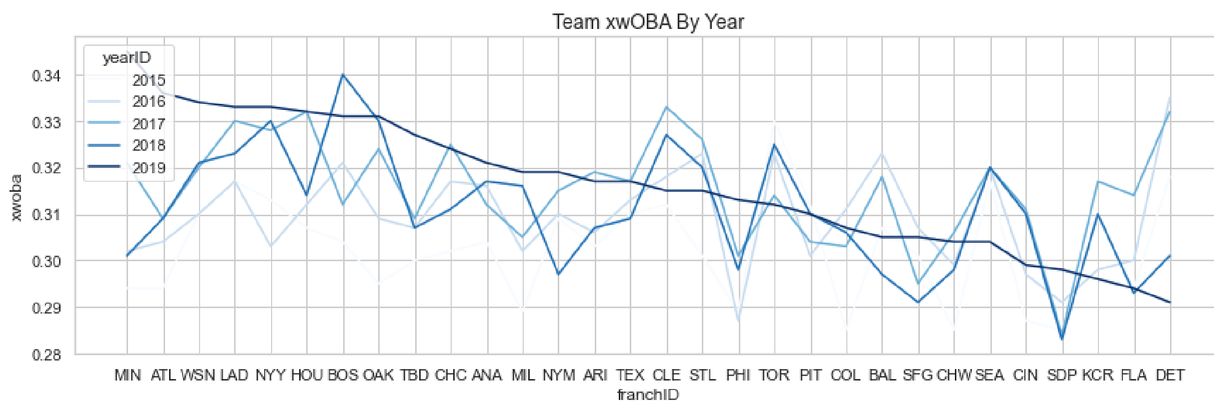
The Boston Red Sox record in 2018 was 108-54

You can clearly see how these advanced stats are great predictors for team performance. Unfortunately, these advanced stats only go back to the year 2015, so they will not be used in my models at this time.

In [67]:
```
xwoba_df.reset_index(inplace=True)
```

In [72]:
```
plt.figure(figsize=(32,14))

plt.subplot(321)
sns.lineplot(xwoba_df.franchID, xwoba_df.xwoba, hue=xwoba_df.yearID, palette='Blu
plt.title('Team xwOBA By Year', fontsize=14)
```

Out[72]:
```
Text(0.5, 1.0, 'Team xwOBA By Year')
```



Vegas Lines

I was able to find the lines set by the bookmakers for MLB season total wins for each team over the past 6 years. I was excited to be able to find this because this allows me to compare the vegas lines vs. the actual outcomes. As you can imagine, this is not something that is readily available by a simple google search because this is like finding the data for a casino and knowing exactly how accurate/profitable each game and table really is. Lets dig in!

In [254]:
```
vegas_odds = pd.read_csv('data/vegas_mlb_seaswins.csv')
```

In [255]:
```
vegas_odds.drop_duplicates(inplace=True)
```

In [256]:
```
vegas_odds.tail()
```

Out[256]:

	year	Team	vegas_win_total
176	2019.000	TBD	85.500
177	2019.000	TEX	70.500
178	2019.000	TOR	76.500
179	2019.000	WSN	88.500
180	nan	NaN	nan

In [257]:
```
# remove last row - should be 180 rows
vegas_odds = vegas_odds.iloc[:-1 , :]
```

In [259]:
```
# rename columns and lowercase all columns
vegas_odds = vegas_odds.rename(columns={'year': 'yearID',
                                        'Team': 'franchID'})
```

In [261]:
```
vegas_odds.set_index(['yearID','franchID'], inplace=True)
```

In [116]:
```
df_teams.set_index(['yearID','franchID'], inplace=True)
```

...

In [262]:
```
# join the vegas odds to the main teams df
vegas_df = df_teams.join(vegas_odds, how='inner', lsuffix='teams_', rsuffix='stat
vegas_df.info()
```

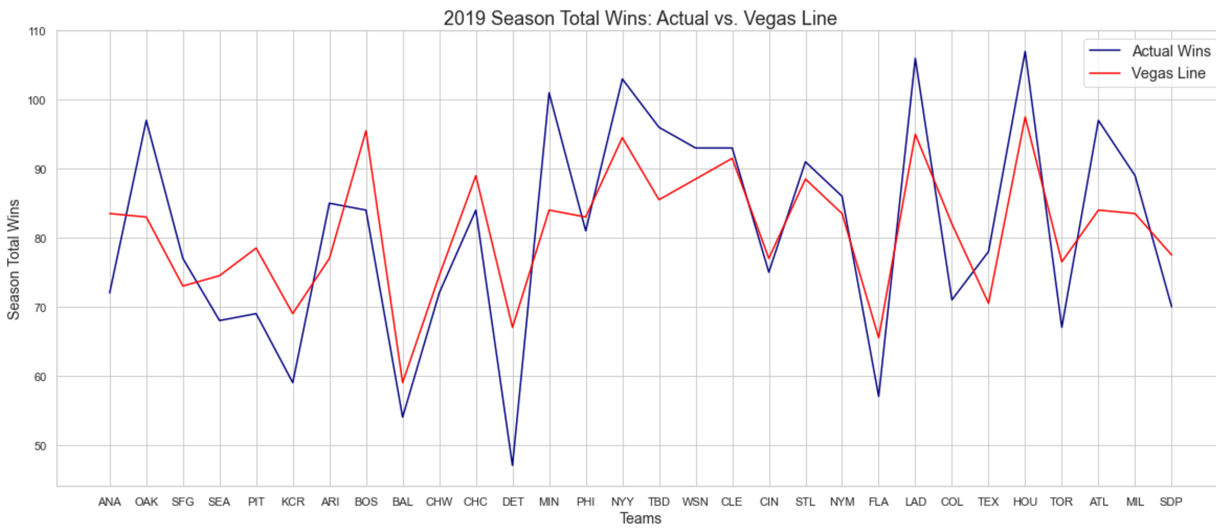
```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 180 entries, (2017.0, LAD) to (2016.0, WSN)
Data columns (total 33 columns):
W                  180 non-null int64
R                  180 non-null int64
AB                 180 non-null int64
H                  180 non-null int64
2B                 180 non-null int64
3B                 180 non-null int64
HR                 180 non-null int64
BB                 180 non-null float64
SO                 180 non-null float64
SB                 180 non-null float64
SF                 180 non-null float64
RA                 180 non-null int64
ER                 180 non-null int64
ERA                180 non-null float64
CG                 180 non-null int64
SHO                180 non-null int64
SV                 180 non-null int64
HA                 180 non-null int64
HRA                180 non-null int64
BBA                180 non-null int64
SOA                180 non-null int64
E                  180 non-null int64
DP                 180 non-null int64
FP                 180 non-null float64
BPF                180 non-null int64
PPF                180 non-null int64
DivWin_Y           180 non-null uint8
WCWin_Y            180 non-null uint8
LgWin_Y            180 non-null uint8
WSWin_Y            180 non-null uint8
run_diff           180 non-null int64
ba                 180 non-null float64
vegas_win_total    180 non-null float64
dtypes: float64(8), int64(21), uint8(4)
memory usage: 42.3+ KB
```

In [264]:
```
vegas_df.reset_index(inplace=True)
```

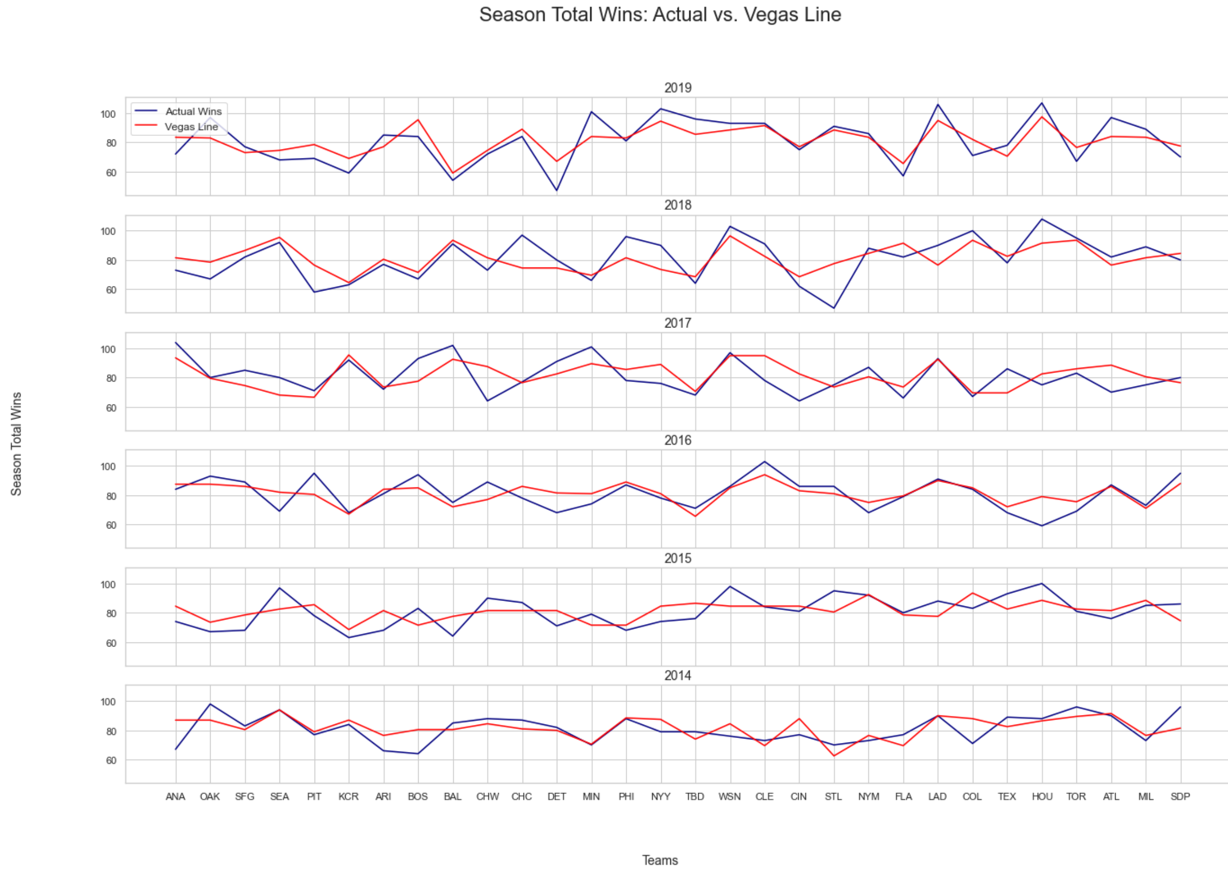
In [265]:
```
vegas_df['yearID'] = pd.to_datetime(vegas_df['yearID'], format='%Y')
```

In [267]:
```
vegas_df_2019 = vegas_df.loc[vegas_df['yearID']=='2019-01-01']
vegas_df_2018 = vegas_df.loc[vegas_df['yearID']=='2018-01-01']
vegas_df_2017 = vegas_df.loc[vegas_df['yearID']=='2017-01-01']
vegas_df_2016 = vegas_df.loc[vegas_df['yearID']=='2016-01-01']
vegas_df_2015 = vegas_df.loc[vegas_df['yearID']=='2015-01-01']
vegas_df_2014 = vegas_df.loc[vegas_df['yearID']=='2014-01-01']
```

In [268]:
```
plt.figure(figsize=(20,8))
X = vegas_df_2019['franchID']
y = vegas_df_2019['W']
z = vegas_df_2019['vegas_win_total']

plt.plot(X, y, color='navy', label='Actual Wins')
plt.plot(X, z, color='red', label='Vegas Line')

plt.xlabel('Teams', fontsize=14)
plt.ylabel('Season Total Wins', fontsize=14)
plt.title('2019 Season Total Wins: Actual vs. Vegas Line', fontsize=18)

plt.legend(fontsize='large')
plt.show()
```



In [322]:
```
fig, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(6, sharex=True, sharey=True, f
fig.suptitle('Season Total Wins: Actual vs. Vegas Line', fontsize=22)
fig.text(0.5,0.04, "Teams", ha="center", va="center", fontsize=14)
fig.text(0.05,0.5, "Season Total Wins", ha="center", va="center", rotation=90, fo

X = vegas_df_2019['franchID']
y = vegas_df_2019['W']
z = vegas_df_2019['vegas_win_total']

w = vegas_df_2018['W']
b = vegas_df_2018['vegas_win_total']

v = vegas_df_2017['W']
c = vegas_df_2017['vegas_win_total']

u = vegas_df_2016['W']
d = vegas_df_2016['vegas_win_total']

t = vegas_df_2015['W']
e = vegas_df_2015['vegas_win_total']

s = vegas_df_2014['W']
f = vegas_df_2014['vegas_win_total']

ax1.plot(X, y, color='navy', label='Actual Wins')
ax1.plot(X, z, color='red', label='Vegas Line')

ax2.plot(X, w, color='navy', label='Actual Wins')
ax2.plot(X, b, color='red', label='Vegas Line')

ax3.plot(X, v, color='navy', label='Actual Wins')
ax3.plot(X, c, color='red', label='Vegas Line')

ax4.plot(X, u, color='navy', label='Actual Wins')
ax4.plot(X, d, color='red', label='Vegas Line')

ax5.plot(X, t, color='navy', label='Actual Wins')
ax5.plot(X, e, color='red', label='Vegas Line')

ax6.plot(X, s, color='navy', label='Actual Wins')
ax6.plot(X, f, color='red', label='Vegas Line')

ax1.set_title('2019', fontsize=14)
ax2.set_title('2018', fontsize=14)
ax3.set_title('2017', fontsize=14)
ax4.set_title('2016', fontsize=14)
ax5.set_title('2015', fontsize=14)
ax6.set_title('2014', fontsize=14)

ax1.legend(fontsize='medium', loc='upper left')
plt.savefig('./images/vegas_accuracy.png')
plt.show();
```



In [323]:
```
# calculate absolute diff wins - vegas line
# calculate percent (diff wins - vegas line)/vegas line
vegas_df['abs_line_accuracy'] = vegas_df['W'] - vegas_df['vegas_win_total']
vegas_df['pdiff_line_accuracy'] = (vegas_df['W'] - vegas_df['vegas_win_total']) /
```

In [324]:
```
# Vegas accuracy by year and team 2014 - 2019
compare_lines = vegas_df.groupby(['yearID','franchID'])['W','vegas_win_total','ab
compare_lines
```

Out[324]:

yearID	franchID	W	vegas_win_total	abs_line_accuracy	pdiff_line_accuracy
2014-01-01	**ANA**	98	87.000	11.000	0.126
	ARI	64	80.500	-16.500	-0.205
	ATL	79	87.500	-8.500	-0.097
	BAL	96	81.500	14.500	0.178
	BOS	71	88.000	-17.000	-0.193
...	**...**	...	...	...	...
2019-01-01	**STL**	91	88.500	2.500	0.028
	TBD	96	85.500	10.500	0.123
	TEX	78	70.500	7.500	0.106
	TOR	67	76.500	-9.500	-0.124
	WSN	93	88.500	4.500	0.051

180 rows × 4 columns

In [329]:
```
# Vegas accuracy by year cumulative of all teams
compare_lines2 = vegas_df.groupby(['yearID'])['W','vegas_win_total','abs_line_acc
compare_lines2['pdiff_line_accuracy'] = (compare_lines2['W'] - compare_lines2['ve
compare_lines2
```

Out[329]:

yearID	W	vegas_win_total	abs_line_accuracy	pdiff_line_accuracy
2014-01-01	2430	2454.500	-24.500	-0.010
2015-01-01	2429	2436.000	-7.000	-0.003
2016-01-01	2427	2436.500	-9.500	-0.004
2017-01-01	2430	2447.500	-17.500	-0.007
2018-01-01	2431	2433.000	-2.000	-0.001
2019-01-01	2429	2431.500	-2.500	-0.001

In [330]:
```
# Vegas accuracy by team cumulative 2014 - 2019
compare_lines3 = vegas_df.groupby(['franchID'])['W','vegas_win_total','abs_line_a
compare_lines3['pdiff_line_accuracy'] = (compare_lines3['W'] - compare_lines3['ve
compare_lines3.sort_values('pdiff_line_accuracy', ascending=False)
```

Out[330]:

franchID	W	vegas_win_total	abs_line_accuracy	pdiff_line_accuracy
HOU	551	508.000	43.000	0.085
MIL	494	464.000	30.000	0.065
NYY	549	524.000	25.000	0.048
CHC	544	524.000	20.000	0.038
OAK	494	476.500	17.500	0.037
CLE	546	527.500	18.500	0.035
LAD	579	560.500	18.500	0.033
ATL	473	459.000	14.000	0.031
MIN	476	462.000	14.000	0.030
TBD	491	478.000	13.000	0.027
STL	538	524.000	14.000	0.027
WSN	546	546.000	0.000	0.000
PIT	490	492.500	-2.500	-0.005
ARI	472	475.000	-3.000	-0.006
KCR	462	469.000	-7.000	-0.015
COL	458	465.000	-7.000	-0.015
NYM	489	497.000	-8.000	-0.016
SEA	484	492.000	-8.000	-0.016
PHI	434	441.500	-7.500	-0.017
BOS	527	540.500	-13.500	-0.025
FLA	424	437.000	-13.000	-0.030
ANA	489	504.000	-15.000	-0.030
TOR	481	496.000	-15.000	-0.030
TEX	473	489.000	-16.000	-0.033
BAL	442	458.000	-16.000	-0.035
CHW	428	451.500	-23.500	-0.052
SFG	473	499.000	-26.000	-0.052
SDP	426	452.000	-26.000	-0.058
CIN	418	453.000	-35.000	-0.077
DET	425	473.500	-48.500	-0.102

As you can see from the above, the bookmakers are EXTREMELY accurate!

With that said, there is value in looking into the teams with the largest differences and analyzing how/why the bookmakers are off on their predictions for these teams.

In [291]:

```
print(df_teams.shape)
print(vegas_df.shape)
```

```
(600, 32)
(180, 37)
```

In [292]:

```
import pickle
df_teams.to_pickle('df_teams.pkl')
vegas_df.to_pickle('vegas_df.pkl')
```

Single Game Results

With our knowledge of season total wins, it is now time to dig into single game results. From the lahman's baseball database, we have every game's stats and results from 2014 - 2019.

In [293]:

```
df_gamelogs = pd.read_csv('data/gamelogs.csv')
df_gamelogs.head()
```

Out[293]:

	date	dayofweek	visiting_team	v_game_number	home_team	h_game_number	v_score	h_
0	20190320	Wed	SEA	1	OAK	1	9	
1	20190321	Thu	SEA	2	OAK	2	5	
2	20190328	Thu	PIT	1	CIN	1	3	
3	20190328	Thu	ARI	1	LAD	1	5	
4	20190328	Thu	COL	1	FLA	1	6	

In [297]:
```
# there were 16 missing values to fill
df_gamelogs['LOG_minutes'].fillna(method='backfill', inplace=True)
df_gamelogs.isna().sum()
```

Out[297]:
```
date              0
dayofweek         0
visiting_team     0
v_game_number     0
home_team         0
h_game_number     0
v_score           0
h_score           0
LOG_inouts        0
LOG_minutes       0
v_AB              0
v_H               0
v_2B              0
v_3B              0
v_HR              0
v_RBI             0
v_SF              0
v_HBP             0
v_BB              0
```

In [298]:
```
df_gamelogs['date'] = pd.to_datetime(df_gamelogs['date'], format='%Y%m%d')
df_gamelogs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14577 entries, 0 to 14576
Data columns (total 46 columns):
date                 14577 non-null datetime64[ns]
dayofweek            14577 non-null object
visiting_team        14577 non-null object
v_game_number        14577 non-null int64
home_team            14577 non-null object
h_game_number        14577 non-null int64
v_score              14577 non-null int64
h_score              14577 non-null int64
LOG_inouts           14577 non-null int64
LOG_minutes          14577 non-null float64
v_AB                 14577 non-null int64
v_H                  14577 non-null int64
v_2B                 14577 non-null int64
v_3B                 14577 non-null object
v_HR                 14577 non-null int64
v_RBI                14577 non-null int64
v_SF                 14577 non-null int64
v_HBP                14577 non-null int64
v_BB                 14577 non-null int64
v_K                  14577 non-null int64
v_SB                 14577 non-null int64
v_GIDP               14577 non-null int64
v_LOB                14577 non-null int64
v_pitchers_used      14577 non-null int64
v_ER                 14577 non-null int64
v_def_assists        14577 non-null int64
v_errors_def         14577 non-null int64
v_doubleplays_def    14577 non-null int64
h_AB                 14577 non-null int64
h_H                  14577 non-null int64
h_2B                 14577 non-null int64
h_3B                 14577 non-null int64
h_HR                 14577 non-null int64
h_RBI                14577 non-null int64
h_SF                 14577 non-null int64
h_HBP                14577 non-null int64
h_BB                 14577 non-null int64
h_K                  14577 non-null int64
h_SB                 14577 non-null int64
h_GIDP               14577 non-null int64
h_LOB                14577 non-null int64
h_pitchers_used      14577 non-null int64
h_ER                 14577 non-null int64
h_def_assists        14577 non-null int64
h_errors_def         14577 non-null int64
h_doubleplays_def    14577 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(40), object(4)
memory usage: 5.1+ MB
```

In [299]:
```
df_gamelogs.describe()
```

Out[299]:

	v_game_number	h_game_number	v_score	h_score	LOG_inouts	LOG_minutes	v_/
count	14577.000	14577.000	14577.000	14577.000	14577.000	14577.000	14577.0
mean	81.484	81.483	4.392	4.514	53.611	186.157	68.8
std	46.756	46.758	3.176	3.122	5.108	28.071	1058.6
min	1.000	1.000	0.000	0.000	27.000	75.000	17.0
25%	41.000	41.000	2.000	2.000	51.000	168.000	32.0
50%	82.000	81.000	4.000	4.000	54.000	183.000	34.0
75%	122.000	122.000	6.000	6.000	54.000	200.000	37.0
max	163.000	163.000	24.000	25.000	114.000	413.000	41567.0

In [300]:
```
df_gamelogs.set_index('date', inplace=True)
```

In [301]:
```
df_gamelogs = df_gamelogs.sort_index()
df_gamelogs
```

Out[301]:

	dayofweek	visiting_team	v_game_number	home_team	h_game_number	v_score	h_score
date							
2014-03-22	Sat	LAD	1	ARI	1	3	1
2014-03-23	Sun	LAD	2	ARI	2	7	5
2014-03-30	Sun	LAD	3	SDP	1	1	3
2014-03-31	Mon	ATL	1	MIL	1	0	2
2014-03-31	Mon	WSN	1	NYM	1	9	7
...	...	...	...	...	...	...	...
2019-09-29	Sun	BAL	162	BOS	162	4	5
2019-09-29	Sun	DET	161	CHW	161	3	5
2019-09-29	Sun	MIN	162	KCR	162	4	5
2019-09-29	Sun	NYY	162	TEX	162	1	6
2019-09-29	Sun	HOU	162	ANA	162	8	5

14577 rows × 45 columns

In [302]:
```
df_gamelogs.to_pickle('df_gamelogs.pkl')
```

Rolling Statistics:

In [303]:
```
df = pd.read_pickle('df_gamelogs.pkl')
df
```

Out[303]:

	dayofweek	visiting_team	v_game_number	home_team	h_game_number	v_score	h_score
date							
2014-03-22	Sat	LAD	1	ARI	1	3	1
2014-03-23	Sun	LAD	2	ARI	2	7	5
2014-03-30	Sun	LAD	3	SDP	1	1	3
2014-03-31	Mon	ATL	1	MIL	1	0	2
2014-03-31	Mon	WSN	1	NYM	1	9	7
...	...	...	...	...	...	...	...
2019-09-29	Sun	BAL	162	BOS	162	4	5
2019-09-29	Sun	DET	161	CHW	161	3	5
2019-09-29	Sun	MIN	162	KCR	162	4	5
2019-09-29	Sun	NYY	162	TEX	162	1	6
2019-09-29	Sun	HOU	162	ANA	162	8	5

14577 rows × 45 columns

In [304]:
```
df.reset_index(inplace=True)
```

In [305]:
```
teams = list(df['visiting_team'].value_counts().index)
teams
```

Out[305]:
```
['MIL',
 'COL',
 'SFG',
 'LAD',
 'NYY',
 'STL',
 'HOU',
 'TOR',
 'CHC',
 'TEX',
 'WSN',
 'BAL',
 'ARI',
 'CHW',
 'ANA',
 'MIN',
 'CIN',
 'PIT',
 'KCR',
```

In [306]:
```
# create blank df with features needed for analysis
new_df = pd.DataFrame(columns = ['date', 'team', 'team_score', 'opponent_score',
       'Hits','RBI','doubles','strikeouts', 'def_assists', 'def_errors', 'def_dou
       'pitchers_used'])
```

In [43]:
```
df.columns
```

Out[43]:
```
Index(['date', 'dayofweek', 'visiting_team', 'v_game_number', 'home_team',
       'h_game_number', 'v_score', 'h_score', 'LOG_inouts', 'LOG_minutes',
       'v_AB', 'v_H', 'v_2B', 'v_3B', 'v_HR', 'v_RBI', 'v_SF', 'v_HBP', 'v_BB',
       'v_K', 'v_SB', 'v_GIDP', 'v_LOB', 'v_pitchers_used', 'v_ER',
       'v_def_assists', 'v_errors_def', 'v_doubleplays_def', 'h_AB', 'h_H',
       'h_2B', 'h_3B', 'h_HR', 'h_RBI', 'h_SF', 'h_HBP', 'h_BB', 'h_K', 'h_SB',
       'h_GIDP', 'h_LOB', 'h_pitchers_used', 'h_ER', 'h_def_assists',
       'h_errors_def', 'h_doubleplays_def'],
      dtype='object')
```

In [307]:
```
v_temp = df.loc[df['visiting_team'] == 'SEA',['date', 'visiting_team', 'v_score',
                                                              'v_2B','v_K','v_def_ass
                                                              'v_H': 'Hits','v_RBI':
                                                              'v_errors_def': 'def_er
v_temp
```

Out[307]:

	date	team	team_score	opponent_score	Hits	RBI	doubles	strikeouts	def_assists	def_
13	2014-03-31	SEA	10	3	11	10	4	11	5	
21	2014-04-01	SEA	8	3	10	8	3	12	14	
38	2014-04-02	SEA	8	2	13	8	2	9	11	
45	2014-04-03	SEA	2	3	6	2	1	9	11	
74	2014-04-05	SEA	3	1	7	3	2	9	7	
...	...	...	...	...	...	...	...	...	...	
14425	2019-09-18	SEA	4	1	7	4	2	10	12	
14436	2019-09-19	SEA	6	5	13	5	1	12	13	
14447	2019-09-20	SEA	3	5	4	3	1	5	9	
14469	2019-09-21	SEA	7	6	11	6	1	10	16	
14477	2019-09-22	SEA	1	2	7	1	3	5	11	

486 rows × 12 columns

In [308]:
```
h_temp = df.loc[df['home_team'] == 'SEA',['date', 'home_team', 'v_score', 'h_scor
                                                          'h_2B','h_K','h_def_ass
                                                          'h_H': 'Hits','h_RBI':
                                                          'h_errors_def': 'def_er
h_temp
```

Out[308]:

	date	team	opponent_score	team_score	Hits	RBI	doubles	strikeouts	def_assists	def_
107	2014-04-08	SEA	3	5	8	5	2	9	10	
123	2014-04-09	SEA	2	0	1	0	0	9	9	
149	2014-04-11	SEA	4	6	11	4	2	12	3	
164	2014-04-12	SEA	3	1	7	1	1	11	9	
178	2014-04-13	SEA	3	0	3	0	0	11	6	
...	...	...	...	...	...	...	...	...	...	
14509	2019-09-25	SEA	3	0	2	0	0	10	8	
14522	2019-09-26	SEA	3	1	4	1	1	10	9	
14534	2019-09-27	SEA	3	4	8	4	3	9	7	
14548	2019-09-28	SEA	1	0	6	0	0	6	11	
14562	2019-09-29	SEA	1	3	7	3	2	7	8	

486 rows × 12 columns

In [309]:
```
#loop through each team, create temp df for when they are home and away, rename c
for team in teams:
    v_temp = df.loc[df['visiting_team'] == team,['date', 'visiting_team', 'v_scor
                                                   'v_2B','v_K','v_def_assists','v_e
                                                   'v_H': 'Hits','v_RBI': 'RBI', 'v_
                                                   'v_errors_def': 'def_errors','v_d
    h_temp = df.loc[df['home_team'] == team,['date', 'home_team', 'v_score', 'h_s
                                              'h_2B','h_K','h_def_assists','h_error
                                              'h_H': 'Hits','h_RBI': 'RBI', 'h_2B':
                                              'h_errors_def': 'def_errors','h_doubl
    new_df = pd.concat([new_df,v_temp,h_temp])
```

In [310]:
```
new_df.shape
```
Out[310]:
```
(29154, 12)
```

In [311]:
```
new_df.head()
```
Out[311]:

	Hits	RBI	date	def_assists	def_doubleplays	def_errors	doubles	opponent_score	pitchers_
62	12	6	2014-04-04	7	2	1	4	2	
71	19	7	2014-04-05	14	2	2	6	6	
90	9	3	2014-04-06	10	0	1	2	0	
104	15	9	2014-04-08	4	0	0	5	4	
118	13	8	2014-04-09	11	1	1	3	4	

In [312]:
```
new_df.sort_values('date', inplace= True)
```

In [313]:
```
new_df = new_df[['date', 'team', 'team_score', 'opponent_score',
         'Hits','RBI','doubles','strikeouts', 'def_assists', 'def_errors', 'def_dou
         'pitchers_used']]
```

In [314]:
```
new_df
```

Out[314]:

	date	team	team_score	opponent_score	Hits	RBI	doubles	strikeouts	def_assists	def_
0	2014-03-22	ARI	1	3	5	1	1	10	10	
0	2014-03-22	LAD	3	1	5	3	2	11	13	
1	2014-03-23	LAD	7	5	13	6	3	7	4	
1	2014-03-23	ARI	5	7	8	5	0	8	15	
2	2014-03-30	SDP	3	1	5	3	0	10	10	
...	...	...	...	...	...	...	...	...	...	
14573	2019-09-29	CHW	5	3	4	4	2	12	15	
14566	2019-09-29	PHI	3	4	13	3	1	10	7	
14563	2019-09-29	ARI	1	0	7	1	2	8	14	
14569	2019-09-29	STL	9	0	9	8	0	7	11	
14568	2019-09-29	LAD	9	0	12	9	1	8	10	

29154 rows × 12 columns

In [315]:
```
# calculate rolling statistics
new_df['Rolling3_runs_scored'] = new_df.groupby('team')['team_score'].transform(l
new_df['Rolling3_runs_against'] = new_df.groupby('team')['opponent_score'].transf
new_df['Rolling3_run_diff'] = new_df['Rolling3_runs_scored'] - new_df['Rolling3_r
new_df['Rolling3_hits'] = new_df.groupby('team')['Hits'].transform(lambda x: x.ro
new_df['Rolling3_RBI'] = new_df.groupby('team')['RBI'].transform(lambda x: x.roll
new_df['Rolling3_doubles'] = new_df.groupby('team')['doubles'].transform(lambda x
new_df['Rolling3_strikeouts'] = new_df.groupby('team')['strikeouts'].transform(la
new_df['Rolling3_def_assists'] = new_df.groupby('team')['def_assists'].transform(
new_df['Rolling3_def_errors'] = new_df.groupby('team')['def_errors'].transform(la
new_df['Rolling3_doubleplays'] = new_df.groupby('team')['def_doubleplays'].transf
new_df['Rolling3_pitchers_used'] = new_df.groupby('team')['pitchers_used'].transf
```

In [316]:
```
# create target column that defines the game result as a 1 or 0
new_df['game_result'] = np.where(new_df['team_score'] - new_df['opponent_score']>
```

In [317]:
```
new_df.fillna(method='backfill', inplace=True)
new_df.isna().sum()
```

Out[317]:
```
date                     0
team                     0
team_score               0
opponent_score           0
Hits                     0
RBI                      0
doubles                  0
strikeouts               0
def_assists              0
def_errors               0
def_doubleplays          0
pitchers_used            0
Rolling3_runs_scored     0
Rolling3_runs_against    0
Rolling3_run_diff        0
Rolling3_hits            0
Rolling3_RBI             0
Rolling3_doubles         0
Rolling3_strikeouts      0
Rolling3_def_assists     0
Rolling3_def_errors      0
Rolling3_doubleplays     0
Rolling3_pitchers_used   0
game_result              0
dtype: int64
```

In [318]:
```
new_df
```

Out[318]:

	date	team	team_score	opponent_score	Hits	RBI	doubles	strikeouts	def_assists	def_
0	2014-03-22	ARI	1	3	5	1	1	10	10	
0	2014-03-22	LAD	3	1	5	3	2	11	13	
1	2014-03-23	LAD	7	5	13	6	3	7	4	
1	2014-03-23	ARI	5	7	8	5	0	8	15	
2	2014-03-30	SDP	3	1	5	3	0	10	10	
...	...	...	...	...	...	...	...	...	...	
14573	2019-09-29	CHW	5	3	4	4	2	12	15	
14566	2019-09-29	PHI	3	4	13	3	1	10	7	
14563	2019-09-29	ARI	1	0	7	1	2	8	14	
14569	2019-09-29	STL	9	0	9	8	0	7	11	
14568	2019-09-29	LAD	9	0	12	9	1	8	10	

29154 rows × 24 columns

In [319]:
```
check = new_df.loc[(new_df['team'] == 'NYM') | (new_df['team'] == 'NYM')]
check.head(20)
```

Out[319]:

	date	team	team_score	opponent_score	Hits	RBI	doubles	strikeouts	def_assists	def_er
4	2014-03-31	NYM	7	9	7	7	0	18	11	
26	2014-04-02	NYM	1	5	3	1	1	13	6	
42	2014-04-03	NYM	2	8	7	2	3	8	11	
53	2014-04-04	NYM	4	3	6	4	1	4	11	
65	2014-04-05	NYM	6	3	6	6	1	10	6	
80	2014-04-06	NYM	1	2	4	1	2	8	9	
108	2014-04-08	NYM	4	0	9	3	1	9	8	
129	2014-04-09	NYM	3	4	6	3	1	9	6	
134	2014-04-10	NYM	6	4	9	6	1	9	12	
153	2014-04-11	NYM	4	5	10	3	3	9	11	
167	2014-04-12	NYM	7	6	11	7	0	14	19	
183	2014-04-13	NYM	2	14	6	1	1	11	9	
189	2014-04-14	NYM	7	3	13	7	2	10	14	
204	2014-04-15	NYM	9	0	12	8	1	10	6	
211	2014-04-16	NYM	5	2	10	5	2	8	10	
235	2014-04-18	NYM	0	6	1	0	0	8	11	
250	2014-04-19	NYM	5	7	12	5	3	8	8	
264	2014-04-20	NYM	4	3	9	4	0	11	13	
279	2014-04-21	NYM	2	0	7	2	1	8	11	
292	2014-04-22	NYM	0	3	4	0	0	4	8	

In [320]:
```
# this is the mets record from 2014 - 2019 to confirm data is still in tact
check.game_result.value_counts()
```

Out[320]:
```
1    489
0    483
Name: game_result, dtype: int64
```

In [321]:
```
new_df.to_pickle('new_df.pkl')
```

In [ ]:

In [ ]: