Project

December 2, 2019

1 Data Science Workshop

2 NBA Free Throws Prediction

In January 2015 a data-set of 600K NBA free-throws was upload to Kaggle by Sebastian-Mantey. The data was scraped from the website ESPN.com which belongs to an entertainment and sports programming network.

Since the data-set was uploaded, 25 kernels were uploaded to Kaggle related to it. Most of the kernels summarize, analyze and visualize the data and do not try to predict anything. However, there are two interesting kernels:

The kernel 'Shooting percentage over time', engages with the questions How does the Free Throw shooting percent develop over time? Does it go down as the game approaches the ending due to higher pressure? Does it go up thanks to players being warmer, or alternatively - better shooters take the ball?

The findings of the analysis are that in the end of every quarter of a game, there is an increase in the number of free-throws and in the free-throws percentage. They also found that most of the throws in the end of every quarter were performed by better players, but still the absolute time of the throw affected the outcome more than how performed it.

Another interesting kernel is 'Pooling Partial Hierarchica via champs throw Free', in which they try to find the best players in the data-set with statistical and probability methods such as complete pooling, and find the probability p for every player to succeed in the free-throw.

Outside Kaggle, we found the article 'Mindfulness and Free Throws'. This article investigate the relationship between mindfulness, preshot routine, and basketball free-throw percentage. The findings suggest that the combination of mindfulness levels, skill level (practice free-throw percentage), and competitive experience (year in school), all contribute to the prediction of competitive free throw percentage.

2.1 Data preparation and cleaning

2.2 Import Python Libraries

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
pd.set_option('display.notebook_repr_html', True)
pd.options.display.width = 50
def _repr_latex_(self):
    return "\centering{%s}" % self.to_latex()

pd.DataFrame._repr_latex_ = _repr_latex_  # monkey patch pandas DataFrame
%matplotlib inline
```

2.3 Reading the Original Dataset

[2]:
```
free_throws_db = pd.read_csv('free_throws.csv')
free_throws_db.drop_duplicates()
free_throws_db.head(10)
```

[2]:

	end_result	game	game_id	period	play	player
0	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 1 of 2	Andrew
1	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 2 of 2	Andrew
2	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 1 of 2	Andrew
3	106 - 114	PHX - LAL	261031013	1	Andrew Bynum misses free throw 2 of 2	Andrew
4	106 - 114	PHX - LAL	261031013	1	Shawn Marion makes free throw 1 of 1	Shawn
5	106 - 114	PHX - LAL	261031013	1	Amare Stoudemire makes free throw 1 of 2	Amare
6	106 - 114	PHX - LAL	261031013	1	Amare Stoudemire makes free throw 2 of 2	Amare
7	106 - 114	PHX - LAL	261031013	2	Leandro Barbosa misses free throw 1 of 2	Leandr
8	106 - 114	PHX - LAL	261031013	2	Leandro Barbosa makes free throw 2 of 2	Leandr
9	106 - 114	PHX - LAL	261031013	2	Lamar Odom makes free throw 1 of 2	Lamar

Describition of dataset: - end_result: host total score - guest total score -

game: host team vs guest team - game_id: id of specific game - period: which quarter - play: who make free throw, make or miss free throw - player: player name - playoffs: whether a playoff game or regular game - score: host team score - guest team score at that time - season: NBA season - shot_made: whether player got the free throw - time: time left in that quarter

```
[3]: print("Number of free throws in database: %d"%(free_throws_db.shape[0]))
print("Number of games in database: {}".format(free_throws_db.game_id.unique().
 ↪size))
print("Games distribution:")
free_throws_db['playoffs'].value_counts()
```

```
Number of free throws in database: 618019
Number of games in database: 12874
Games distribution:
```

```
[3]: regular     575893
playoffs     42126
Name: playoffs, dtype: int64
```

2.4 Collecting more data from internet

In order to expand our dataset, we decided to use an open source python library PandasBasketball, and use a webscrapper in order to get more players stats from https://www.basketball-reference.com website

Example of basketball-reference NBA player stats webpage:

Totals Share & more ▼ Glossary

Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
2003-04	19	CLE	NBA	SG	79	79	3122	622	1492	.417	63	217	.290	559	1275	.438	.438	347	460	.754	99	333	432	465	130	58	273	149	1654
2004-05 ★	20	CLE	NBA	SF	80	80	**3388**	**795**	1684	.472	108	308	.351	687	1376	.499	.504	477	636	.750	111	477	588	577	177	52	262	146	2175
2005-06 ★	21	CLE	NBA	SF	79	79	3361	875	1823	.480	127	379	.335	748	1444	.518	.515	601	814	.738	75	481	556	521	123	66	260	181	2478
2006-07 ★	22	CLE	NBA	SF	78	78	**3190**	772	1621	.476	99	310	.319	673	1311	.513	.507	489	701	.698	83	443	526	470	125	55	250	171	2132
2007-08 ★	23	CLE	NBA	SF	75	74	3027	**794**	1642	.484	113	359	.315	681	1283	.531	.518	549	771	.712	133	459	592	539	138	81	255	165	2250
2008-09 ★	24	CLE	NBA	SF	81	81	3054	789	1613	.489	132	384	.344	657	1229	.535	.530	**594**	762	.780	106	507	613	587	137	93	241	139	2304
2009-10 ★	25	CLE	NBA	SF	76	76	2966	768	1528	.503	129	387	.333	639	1141	.560	.545	593	773	.767	71	483	554	651	125	77	261	119	2258
2010-11 ★	26	MIA	NBA	SF	79	79	3063	**758**	1485	.510	92	279	.330	666	1206	.552	.541	503	663	.759	80	510	590	554	124	50	284	163	2111
2011-12 ★	27	MIA	NBA	SF	62	62	2326	621	1169	.531	54	149	.362	**567**	1020	.556	.554	387	502	.771	94	398	492	387	115	50	213	96	1683
2012-13 ★	28	MIA	NBA	PF	76	76	2877	**765**	1354	.565	103	254	.406	**662**	1100	.602	.603	403	535	.753	97	513	610	551	129	67	226	110	2036
2013-14 ★	29	MIA	NBA	PF	77	77	2902	767	1353	.567	116	306	.379	651	1047	.622	.610	439	585	.750	81	452	533	488	121	26	270	126	2089
2014-15 ★	30	CLE	NBA	SF	69	69	2493	624	1279	.488	120	339	.354	504	940	.536	.535	375	528	.710	51	365	416	511	109	49	272	135	1743
2015-16 ★	31	CLE	NBA	SF	76	76	2709	737	1416	.520	87	282	.309	**650**	1134	.573	.551	359	491	.731	111	454	565	514	104	49	249	143	1920
2016-17 ★	32	CLE	NBA	SF	74	74	2794	736	1344	.548	124	342	.363	612	1002	.611	.594	358	531	.674	97	542	639	646	92	44	303	134	1954
2017-18 ★	33	CLE	NBA	PF	**82**	82	**3026**	**857**	1580	.542	149	406	.367	708	1174	.603	.590	388	531	.731	97	612	709	747	116	71	347	136	**2251**
2018-19 ★	34	LAL	NBA	SF	55	55	1937	558	1095	.510	111	327	.339	447	768	.582	.560	278	418	.665	57	408	465	454	72	33	197	94	1505
2019-20	35	LAL	NBA	PG	15	15	524	141	290	.486	28	81	.346	113	209	.541	.534	63	89	.708	15	101	116	**169**	19	9	51	22	373
Career			**NBA**		**1213**	**1212**	**46759**	**11979**	**23768**	**.504**	**1755**	**5109**	**.344**	**10224**	**18659**	**.548**	**.541**	**7203**	**9790**	**.736**	**1458**	**7538**	**8996**	**8831**	**1956**	**930**	**4214**	**2229**	**32916**
11 seasons		**CLE**	**NBA**		**849**	**848**	**33130**	**8369**	**17022**	**.492**	**1251**	**3713**	**.337**	**7118**	**13309**	**.535**	**.528**	**5130**	**6998**	**.733**	**1034**	**5156**	**6190**	**6228**	**1376**	**695**	**2973**	**1618**	**23119**
4 seasons		**MIA**	**NBA**		**294**	**294**	**11168**	**2911**	**5361**	**.543**	**365**	**988**	**.369**	**2546**	**4373**	**.582**	**.577**	**1732**	**2285**	**.758**	**352**	**1873**	**2225**	**1980**	**489**	**193**	**993**	**495**	**7919**
2 seasons		**LAL**	**NBA**		**70**	**70**	**2461**	**699**	**1385**	**.505**	**139**	**408**	**.341**	**560**	**977**	**.573**	**.555**	**341**	**507**	**.673**	**72**	**509**	**581**	**623**	**91**	**42**	**248**	**116**	**1878**

And our code to extract from website html our data

[4]:
```python
from tools import get_player_stats
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module='BeatifulSoup')
dataFrame = get_player_stats("Lebron James")
print(dataFrame.columns)
dataFrame.head(20)
```

```
Index(['Season', 'Age', 'Tm', 'Lg', 'Pos', 'G',
       'GS', 'MP', 'FG', 'FGA', 'FG%', '3P',
       '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%',
       'FT', 'FTA', 'FT%', 'ORB', 'DRB', 'TRB',
       'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS',
       'Height', 'Weight', 'ShootingHand',
       'draftRank'],
      dtype='object')
```

[4]:

	Season	Age	Tm	Lg	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P
0	2003-04	19	CLE	NBA	SG	79	79	3122	622	1492	.417	63	217	.290	55
1	2004-05	20	CLE	NBA	SF	80	80	3388	795	1684	.472	108	308	.351	68
2	2005-06	21	CLE	NBA	SF	79	79	3361	875	1823	.480	127	379	.335	74
3	2006-07	22	CLE	NBA	SF	78	78	3190	772	1621	.476	99	310	.319	67
4	2007-08	23	CLE	NBA	SF	75	74	3027	794	1642	.484	113	359	.315	68
5	2008-09	24	CLE	NBA	SF	81	81	3054	789	1613	.489	132	384	.344	65
6	2009-10	25	CLE	NBA	SF	76	76	2966	768	1528	.503	129	387	.333	63
7	2010-11	26	MIA	NBA	SF	79	79	3063	758	1485	.510	92	279	.330	66
8	2011-12	27	MIA	NBA	SF	62	62	2326	621	1169	.531	54	149	.362	56
9	2012-13	28	MIA	NBA	PF	76	76	2877	765	1354	.565	103	254	.406	66
10	2013-14	29	MIA	NBA	PF	77	77	2902	767	1353	.567	116	306	.379	65
11	2014-15	30	CLE	NBA	SF	69	69	2493	624	1279	.488	120	339	.354	50
12	2015-16	31	CLE	NBA	SF	76	76	2709	737	1416	.520	87	282	.309	65
13	2016-17	32	CLE	NBA	SF	74	74	2794	736	1344	.548	124	342	.363	61
14	2017-18	33	CLE	NBA	PF	82	82	3026	857	1580	.542	149	406	.367	70
15	2018-19	34	LAL	NBA	SF	55	55	1937	558	1095	.510	111	327	.339	44
16	2019-20	35	LAL	NBA	PG	19	19	661	187	375	.499	40	111	.360	14
17	Career			NBA		1217	1216	46896	12025	23853	.504	1767	5139	.344	10

Columns used for this database for each player: - Position : The most common position for the player over his seasons. - FG% - 3P% - FT% - Height - Weight - ShootingHand - draftRank

2.5 We merged both the datasets according our collected data from internet.

Some players stats had been inserted manually because some bugs found on PandasBasketball library

[5]:
```python
database_p1 = pd.read_csv("merged_db_part1.csv")
database_p2 = pd.read_csv("merged_db_part2.csv")

database = pd.concat([database_p1,database_p2])
database = database.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])
# Drop duplicated rows
database.drop_duplicates()
database.head(5)
```

[5]:

	end_result	game	game_id	period	play	player
0	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 1 of 2	Andrew By
1	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 2 of 2	Andrew By
2	106 - 114	PHX - LAL	261031013	1	Andrew Bynum makes free throw 1 of 2	Andrew By
3	106 - 114	PHX - LAL	261031013	1	Andrew Bynum misses free throw 2 of 2	Andrew By
4	106 - 114	PHX - LAL	261031013	1	Shawn Marion makes free throw 1 of 1	Shawn Mar

2.6 Specifing Data Types

[6]:
```
binary_variables = ['shot_made', 'playoffs', 'ShootingHand']
categorical_variables = ['end_result', 'game', 'game_id', 'period', 'play',␣
↪'player', 'season', 'Pos']
numeric_variables = ['score','time','FG%','2P%', '3P%', 'FT%', 'Height',␣
↪'Weight', 'draftRank']
```

[7]:
```
database.count()
```

[7]:
```
end_result      618019
game            618019
game_id         618019
period          618019
play            618019
player          618019
playoffs        618019
score           618019
season          618019
shot_made       618019
time            618019
FG%             618019
2P%             618019
3P%             612941
FT%             618019
Height          618019
Weight          618019
draftRank       579587
Pos             618019
ShootingHand    618019
dtype: int64
```

We can see, we have only two columns with missing data. First - the draftRank column. This values are missing because the players performed the free throw didn't have a draft rank and not because we couldn't collect the data. Second - 3P% has missing values since there are players that have never throws a 3-pointer.

2.7 Analyzing the data

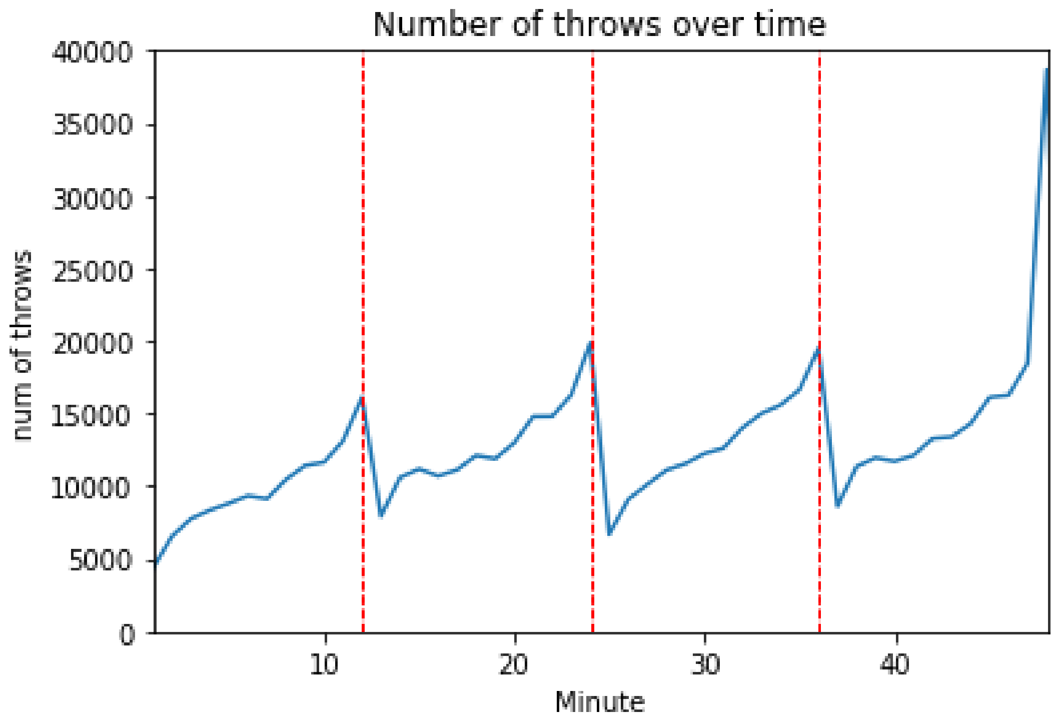
2.7.1 Analyzing the number of throws troughout the game

We would like to show the free throws distribution throughout the game time, in our current dataset, the time column represents the time left in that quarter and the period column represents the quarter of the game, so we'll add a new column to our data which calculate the absolute time in the game that the throw was made.

[8]:
```
database['minute'] = database.time.apply(lambda x: int(x[:len(x)-3]))
database['sec'] = database.time.apply(lambda x: int(x[len(x)-2:]))
database['abs_min'] = 12 - database['minute']+12*(database.period -1)
database['abs_time'] = 60*(database.abs_min-1) + 60 - database['sec']

#  counting the num of throws, and success throws precentage per minute
minutes = range(int(max(database.abs_min)))
total_throws = []
success_throws = []
success_precentage = []

def count_throws(database,minute):
    made = len(database[(database.abs_min == minute) & (database.shot_made ==␣
 ↪1)])
    success_throws.append(made)
    total = len(database[database.abs_min == minute])
    total_throws.append(total)
    if total == 0:
        precentage = 0.0
    else:
        precentage = made/total
    success_precentage.append(precentage)

for minute in minutes:
    count_throws(database,minute)
```

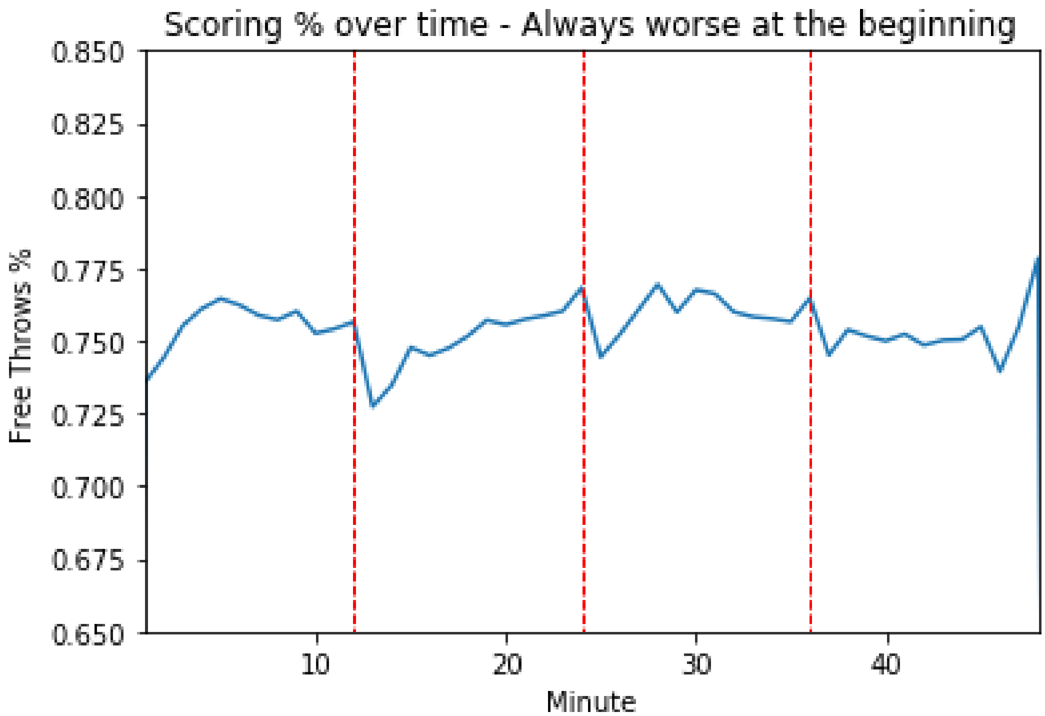
[9]:
```
# Number of throws over time
plt.plot(minutes,total_throws)
plt.title('Number of throws over time')
plt.xlim([1,48])
plt.ylim([0, 40000])
plt.plot([12,12],[0,40000], '--', linewidth = 1, color = 'r')
plt.plot([24,24],[0,40000], '--', linewidth = 1, color = 'r')
plt.plot([36,36],[0,40000], '--', linewidth = 1, color = 'r')
plt.plot([48,48],[0,40000], '--', linewidth = 1, color = 'r')
plt.xlabel('Minute')
plt.ylabel('num of throws')
```

[9]:
```
Text(0, 0.5, 'num of throws')
```



[10]:
```
# Success throws precentage over time
plt.plot(minutes,success_precentage)
plt.title('Scoring % over time - Always worse at the beginning')
plt.xlim([1,48])
plt.ylim([0.65,0.85])
plt.plot([12,12],[0,1], '--', linewidth = 1, color = 'r')
plt.plot([24,24],[0,1], '--', linewidth = 1, color = 'r')
plt.plot([36,36],[0,1], '--', linewidth = 1, color = 'r')
plt.plot([48,48],[0,1], '--', linewidth = 1, color = 'r')
plt.xlabel('Minute')
plt.ylabel('Free Throws %')
```

[10]:
```
Text(0, 0.5, 'Free Throws %')
```



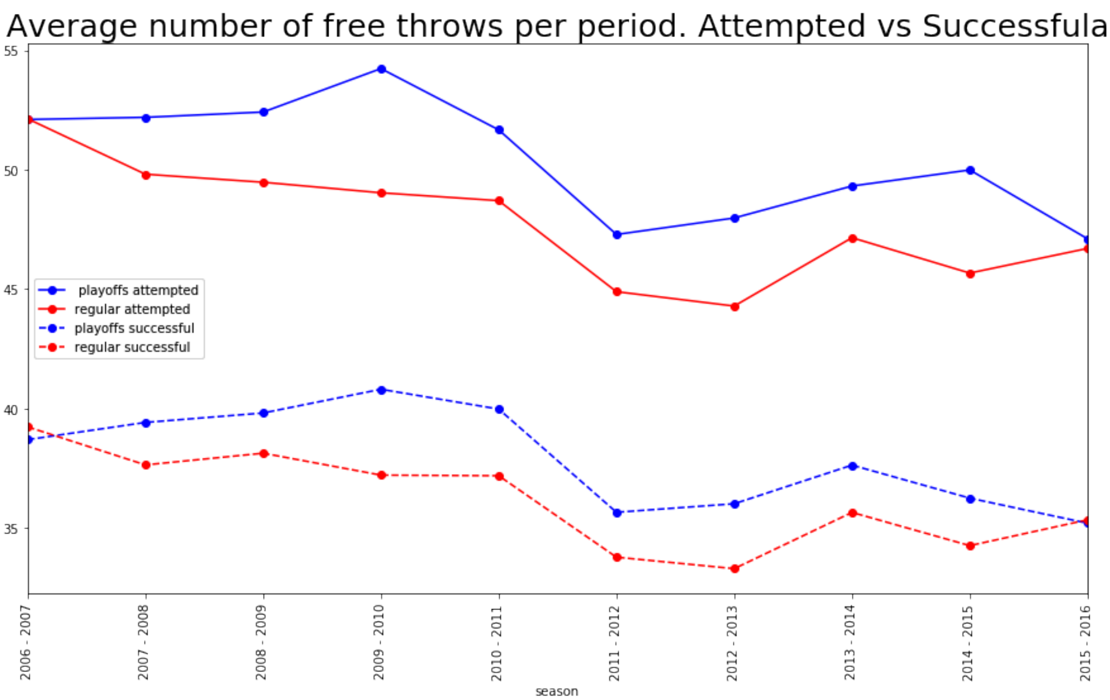
From the plots we can observe that at the begining of every querter, both the number of free throws and the success precentage drops. Moreover, at the end of a quarter, and especially at the end of the game, both plots increase. We can explain this behaivor, as basketball rules when a team made more than 5 fouls, every another foul made in that quarter will be penalty with a free throw.

2.7.2 Analyzing the number of throws troughout the game

We want to see how the average of free throws attempted and succeed in games over our differents season in our dataset are distributed.

[11]:
```
shot_attempted_per_game = database.groupby(["season", "playoffs"])['shot_made'].
    ↪count().unstack()
shot_made_per_game = database.groupby(["season", "playoffs"])['shot_made'].
    ↪sum().unstack()

# this has to be divided by the number of games for each season to get an␣
    ↪average
number_of_games=database.groupby(["season", "playoffs"])['game_id'].nunique().
    ↪unstack()

average_shot_made_per_game = shot_made_per_game/number_of_games
average_shot_attempted_per_game = shot_attempted_per_game/number_of_games
```

```
f, (ax1) = plt.subplots(figsize=(18,18))
first=average_shot_attempted_per_game.plot(ax=ax1, marker='o', figsize=(15,8),␣
↪xticks=range(10), color=['b','r'], rot=90)
second=average_shot_made_per_game.plot(ax=ax1, marker='o', linestyle='--',␣
↪figsize=(15,8), xticks=range(10), color=['b','r'], rot=90)
ax1.set_title('Average number of free throws per period. Attempted vs␣
↪Successfula', size=25)
legend=plt.legend((' playoffs attempted','regular attempted','playoffs␣
↪successful','regular successful'), loc=6)
ax1.add_artist(legend)
plt.show()
```



We can see that are not very big differences over the seasons at both attempted and successful shots. But there is a clear difference between the amount of made shots and and successful shots at playoffs and regular season.

[12]:
```
made_shot_perc , missed_shot_perc = database['shot_made'].
↪value_counts(normalize=True) * 100
print("Made shots percentage in dataset: %.2f"%(made_shot_perc))
print("Missed shots percentage in dataset: %.2f"%(missed_shot_perc))
```

```
Made shots percentage in dataset: 75.68
Missed shots percentage in dataset: 24.32
```

2.7.3 The correlation between the player's draft rank and the free-throw result

We wanted to check if there is a conection between the draft rank of the player performing the shot and the result of the shot. As we saw at the data description, we have some missing values in the draftRank column, and it was because the player throwing had no draft rank.

As we can see below, the majoraty of the data in this column is avalable so we will ommit throws that were made by a players with no draft rank.

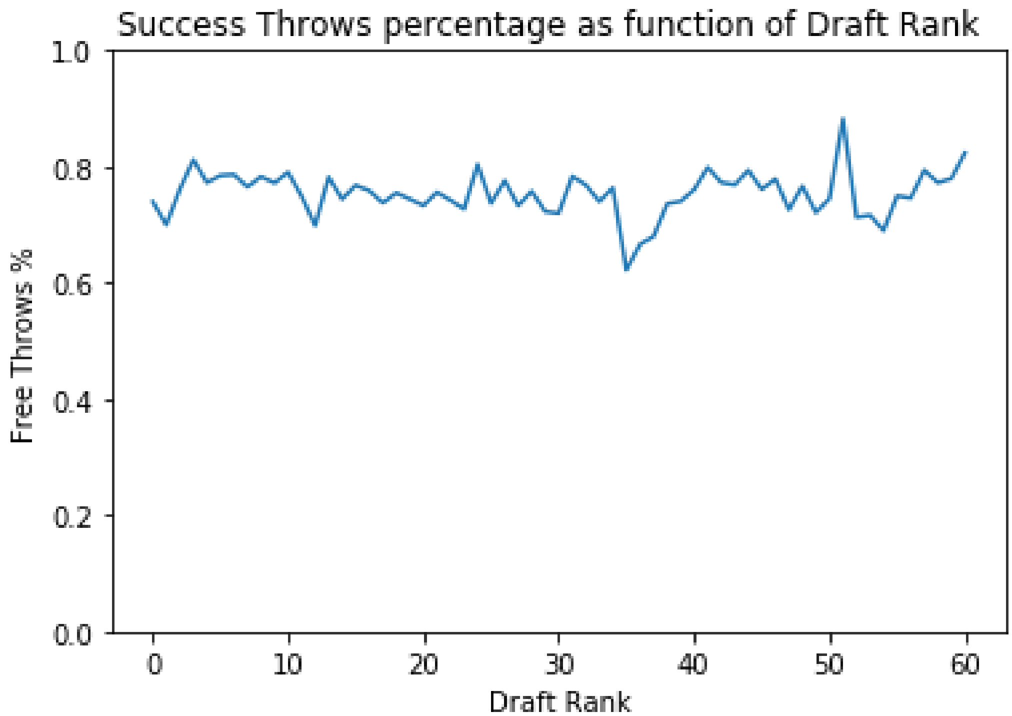
[13]:
```
# the percentage of the data that has draftRank:
print(database['draftRank'].count()/database['game'].count())
```

```
0.9378142095954979
```

[14]:
```
# Function that replace the field "undrafted" with a 0 in draftRank column

database['draftRank'] = database['draftRank'].replace(np.nan, 0)
database['draftRank'] = database['draftRank'].replace("undrafted", 0)
# pd.to_numeric(free_throws_db['draftRank'])
database['draftRank'] = database.draftRank.apply(lambda x: int(float(x)))

np.nanmax(database['draftRank'])

ranks = range(int(np.nanmax(database['draftRank']))+1)
success_precentage_by_rank = []

def throws_per_rank(database,rank):
    total = len(database[database.draftRank == rank])
    if total == 0:
        precentage = 0.0
    else:
        made = len(database[(database.draftRank == rank) & (database.shot_made␣
 ↪== 1)])
        precentage = made/total
    success_precentage_by_rank.append(precentage)

for rank in ranks:
    throws_per_rank(database,rank)
```

[15]:
```
# Success throws precentage over time
plt.plot(list(ranks),success_precentage_by_rank)
plt.title('Success Throws percentage as function of Draft Rank  ')
plt.ylim([0,1])
plt.xlabel('Draft Rank')
plt.ylabel('Free Throws %')
```

[15]:
```
Text(0, 0.5, 'Free Throws %')
```

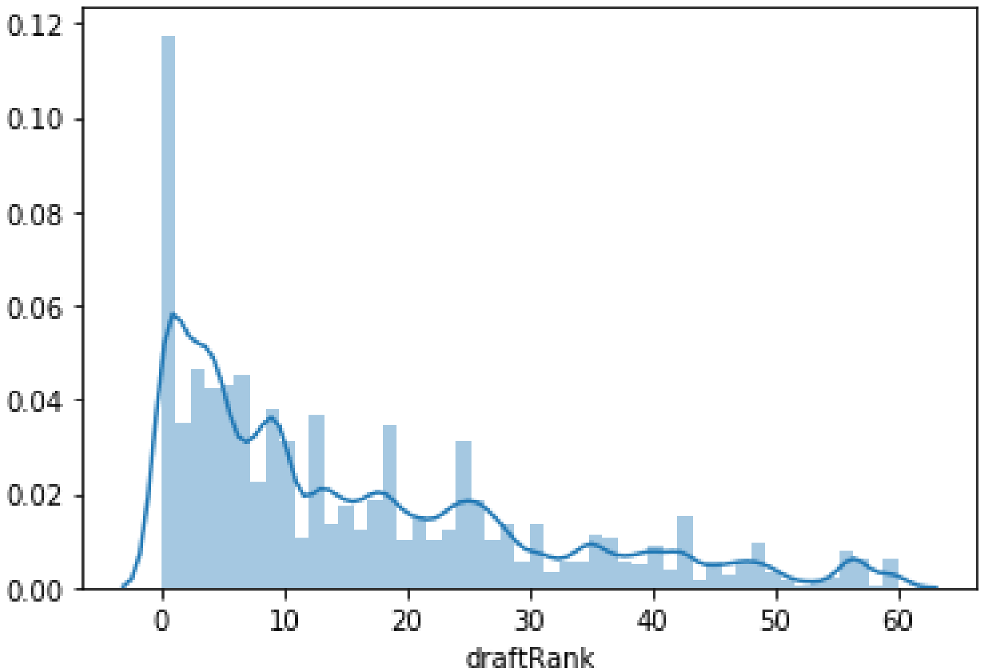


As we see, there is no special trend in the graph (the FT% is between 0.7 to 0.9 for all ranks). We expected that the higher-ranked players would have better performence but we can't conclude it from the data.

We have to take under consideration that the draft-rank data is not uniformly distributed, as we can see in the graph below, and is biased toward small values (the value 0 represents missing values, and in our case players who had no draft rank), so the barplot above needs to be normalized.

[16]:
```
sns.distplot(database['draftRank'])
```

[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1168be250>



2.7.4 The correlation between the score difference and the free-throw result

We will add a column calculating the differance. We assume that if the difference is small in the time of the shot, the player would be more stressed and his shooting precentage would drop.

[17]:
```
database['scores'] = database.score.replace(' - ', '-').apply(lambda x: x.
 ↪split('-'))
database['scoreDif'] = database.scores.apply(lambda x: int(x[1])-int(x[0]))
print(database['scoreDif'])
```

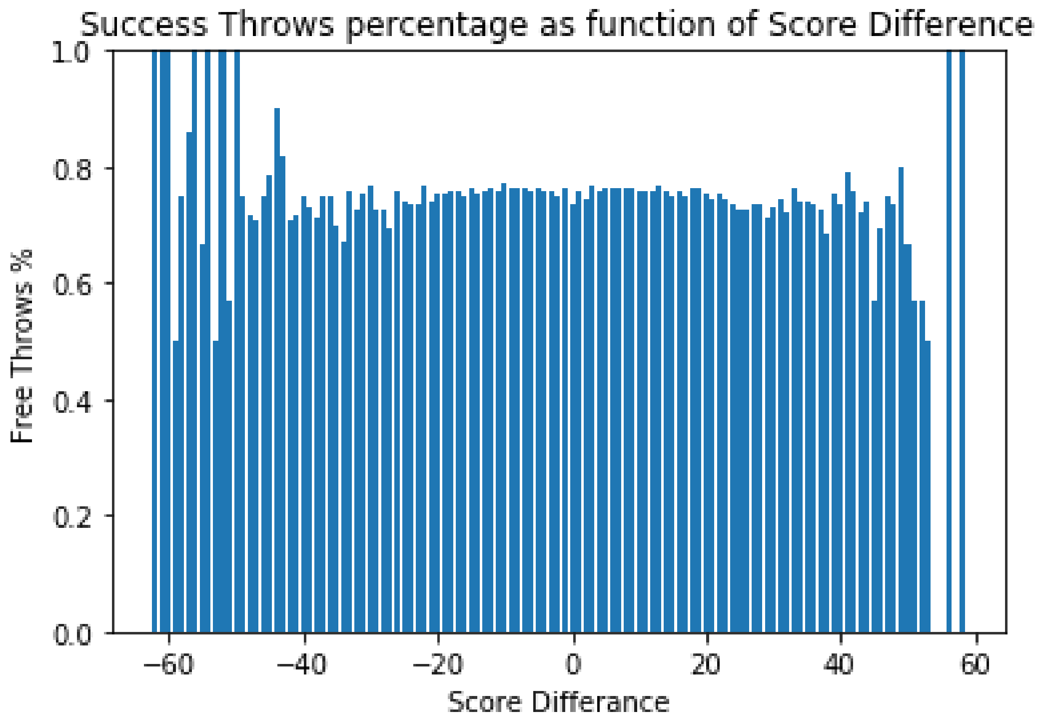
```
0          1
1          2
2         -6
3         -6
4         -9
          ..
309005    13
309006    14
309007    15
309008    15
309009    14
Name: scoreDif, Length: 618019, dtype: int64
```

[18]:
```
difs = range(int(np.min(database['scoreDif']))+1,int(np.
 ↪max(database['scoreDif']))+1)
```

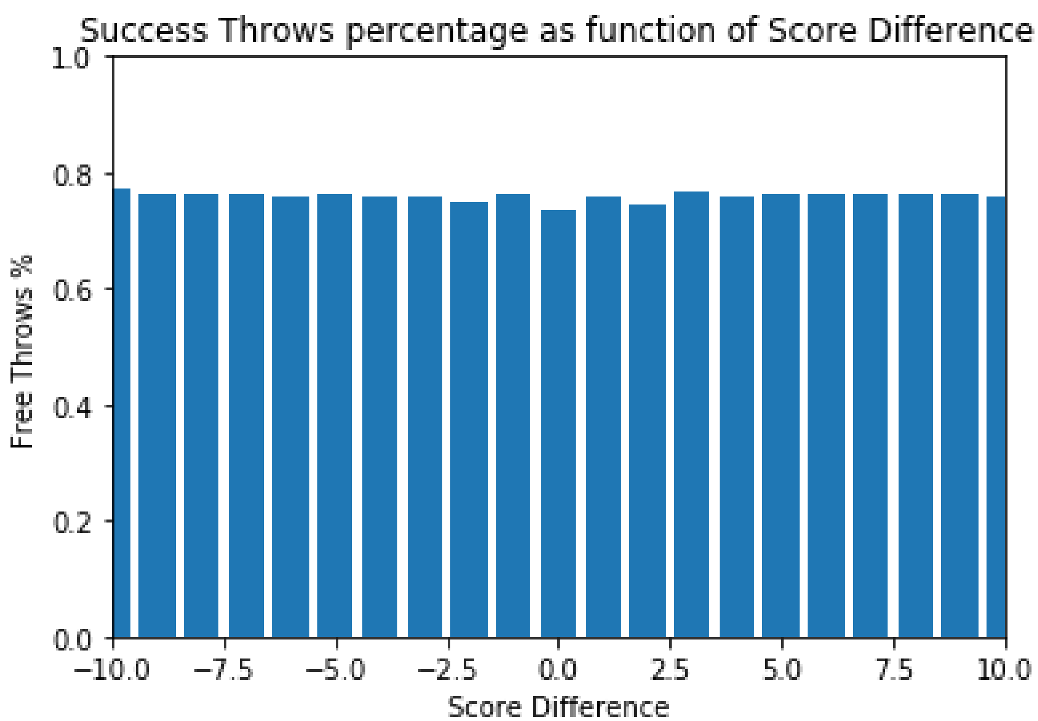
```
success_precentage_by_scoreDif = []

def throws_per_dif(database,dif):
    total = len(database[database.scoreDif == dif])
    if total == 0:
        precentage = 0.0
    else:
        made = len(database[(database.scoreDif == dif) & (database.shot_made ==␣
 ↪1)])
        precentage = made/total
    success_precentage_by_scoreDif.append(precentage)

for dif in difs:
    throws_per_dif(database,dif)
```

[19]:

```
# Success throws precentage over time
plt.bar(list(difs),success_precentage_by_scoreDif)
plt.title('Success Throws percentage as function of Score Difference')
plt.ylim([0,1])
plt.xlabel('Score Differance')
plt.ylabel('Free Throws %')
plt.show()
plt.bar(list(difs),success_precentage_by_scoreDif)
plt.title('Success Throws percentage as function of Score Difference')
plt.ylim([0,1])
plt.xlabel('Score Difference')
plt.ylabel('Free Throws %')
plt.xlim(-10,10)
```



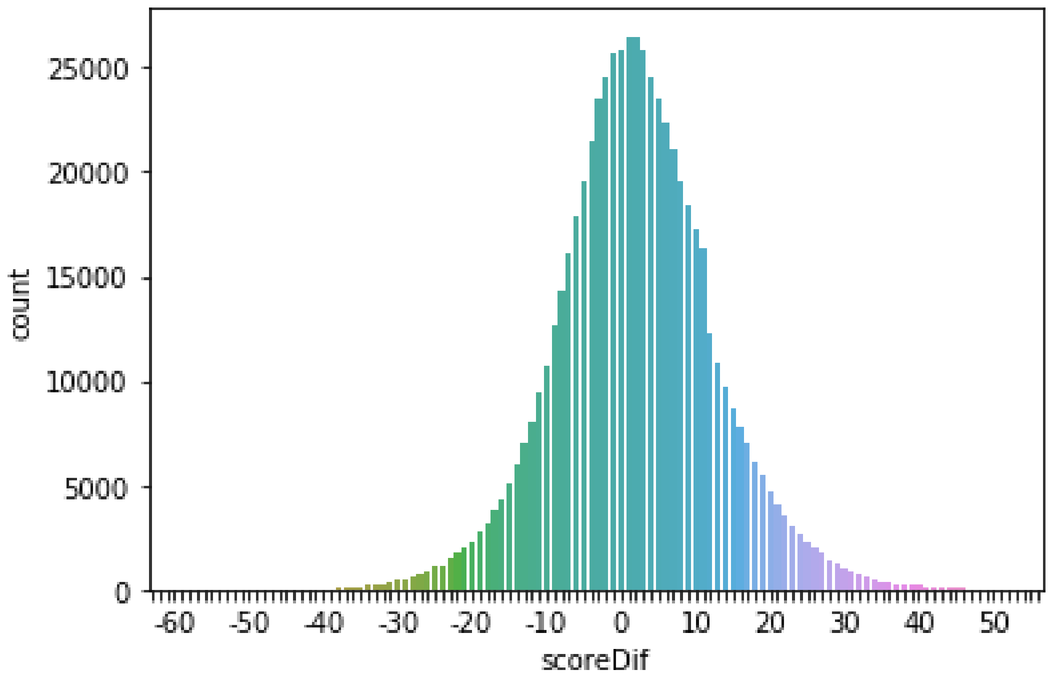
[19]: (-10, 10)



At the `Success Throws percentage as function of Score Difference' plot, it seems that our hypotesis that as long as the score difference gets bigger, so as the free-throw success precentaege is partly true. We can see a trend in the graph but it is not continuous.

Furthermore, here too we have to relate to the fact that we have much more samples with low difference than high difference. As shown at graph below.

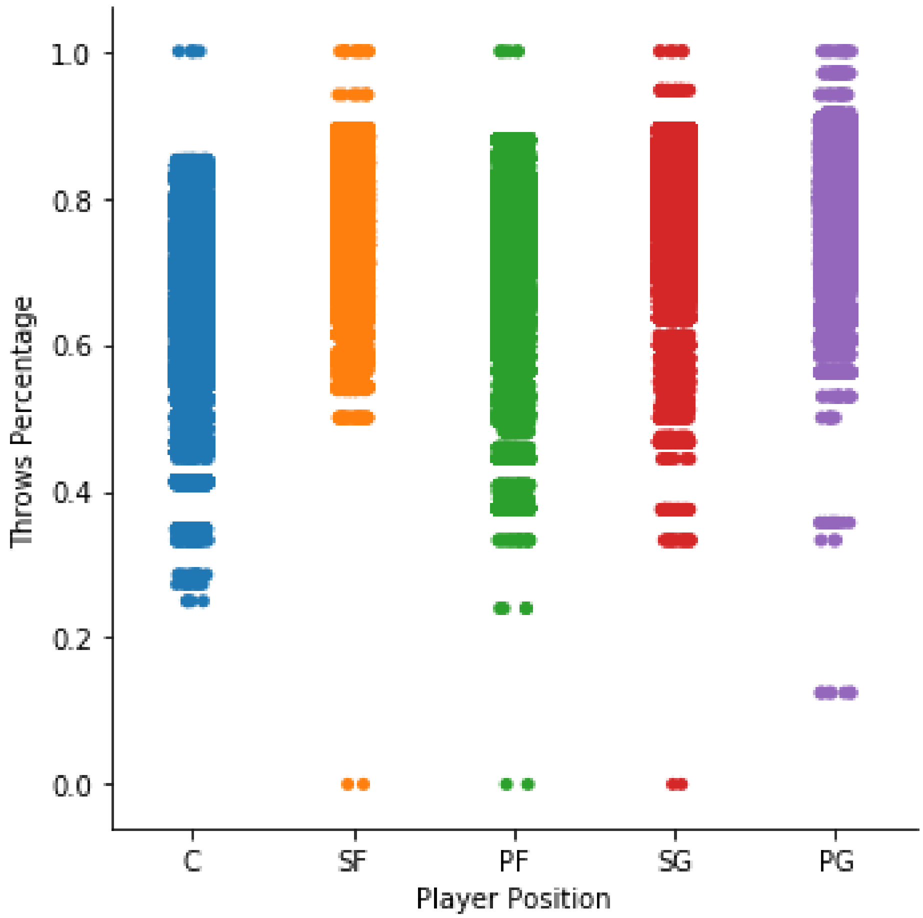
[20]:
```
plot_ = sns.countplot(x='scoreDif', data=database)
for ind, label in enumerate(plot_.get_xticklabels()):
    if (ind-3) % 10 == 0:  # every 10th label is kept
        label.set_visible(True)
    else:
        label.set_visible(False)
plt.show()
```



2.7.5 The correlation between the player position and the free-throw result

We want to analyze if the ``Position'' feature of each player could be a helpful feature that may help our prediction model. Above we can see a plot showing the distribution of the FT% over the different positions.

[21]:
```
ax2 = sns.catplot(x="Pos", y="FT%", data=database);
#ax2 = sns.catplot(x="Pos", y="FT%",  kind="swarm", data=database);
ax2.set(xlabel='Player Position', ylabel='Throws Percentage')
plt.show()
```



Average FT% at each position

[22]:
```
database.groupby('Pos').agg({'FT%':'mean'})
```

[22]:

Pos	FT%
C	0.675366
PF	0.724143
PG	0.805380
SF	0.777317
SG	0.802994

We can see that Centers players owns the worst percentage.

Point Guards and Shouting Guards players holds the best percentage over all the positions.

Next, we will check how well our dataset represents the general population of NBA Players. 1. MSE of the FT% players achieved in their carrers and their FT% we have in the dataset.

[23]:
```
def Dataset_FT_Percentage_per_player(players, database):
    FT_dict = dict()
    for player in players:
        shots_made = len(database[(database.player == player) & (database.
 ↪shot_made == 1)])
        shots_total = len(database[(database.player == player)])
        FT_dict[player] = shots_made/shots_total
    return FT_dict
```

[24]:
```
def Overall_FT_Percentage_per_player(players, database):
    FT_dict_2 = dict()
    df = database.groupby("player")["FT%"].unique()
    FT_dict_2 = df.to_dict()
    return FT_dict_2
```

[25]:
```
players = database["player"].unique()
dataset_percentage_dict = Dataset_FT_Percentage_per_player(players,database)
overall_percentage_dict = Overall_FT_Percentage_per_player(players, database)
```

[26]:
```
MSE = 0
for player in players:
    MSE+=␣
 ↪((overall_percentage_dict[player][0]-dataset_percentage_dict[player])**2)
MSE = MSE/len(players)
print("The MSE between whole carrer FT%% and FT%% calculated from dataset for␣
 ↪the players in our db is: %f"%(MSE))
```

```
The MSE between whole carrer FT% and FT% calculated from dataset for the players
in our db is: 0.003852
```

2.7.6 Our goal

We'll try to explore two options for prediction:

- Offline prediction: try to predict the players succes percentage (for all kinds of throws) for the current season based on his past performance and give the coach an insight about the player.
- Online prediction : try to predict whether a player will hit in a certain throw.

2.7.7 What's next?

- Extract more statistical analysis on our current data, find corralation between more than two parameters.
- Pre-process our data in order to create an initial ML model.
- Achieve good results on our ML model
- Try to reduce our problem dimension/features by analyzing which features have big influence.

- ``Engineer'' more strong features in order to achieve dimension reduction.
- Build new ML model and achieve good results, again! YEY!
- Explain our ML model results, when it fails and why

```
[ ]:
```