Basketball_Markets

September 8, 2020

1 Imports

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
```

```
/Users/ciro/anaconda3/lib/python3.7/site-
packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is
deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1.1 Magic commands

[2]:
```
%config InlineBackend.figure_format ='retina'
```

[7]:
```
# from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = "all"
```

[3]:
```
!ls *.csv #check what files are available with .csv
```

```
basic_per_game_player_stats_2013_2018.csv
df_Basketball.csv
```

[4]:
```
df = pd.read_csv('basic_per_game_player_stats_2013_2018.csv')
pd.set_option('display.max_columns', None)
```

[5]:
```
df = df.rename(columns={'Unnamed: 6': 'Location','Unnamed: 8':'Result'})
```

[6]:
```
# replace NAN values in Location column by 'vs' meaning home game
df.Location = df.Location.fillna('vs')
```

[147]:
```
df.head()
```

[147]:

	Rk	Player	Age	Pos	Date	Tm	Location	Opp	\
0	1	Devin Booker\bookede01	20-145	G	2017-03-24	PHO	@	BOS	
1	2	Carmelo Anthony\anthoca01	29-240	F	2014-01-24	NYK	vs	CHA	
2	3	LeBron James\jamesle01	29-063	F-G	2014-03-03	MIA	vs	CHA	

```
3   4      Kobe Bryant\bryanko01  37-234  G-F  2016-04-13  LAL       vs  UTA
4   5     James Harden\hardeja01  28-157    G  2018-01-30  HOU       vs  ORL

  Result  GS  MP  FG  FGA    FG%  2P  2PA    2P%  3P  3PA    3P%  FT  FTA  \
0      L   1  45  21   40  0.525  17   29  0.586   4   11  0.364  24   26
1      W   1  39  23   35  0.657  17   24  0.708   6   11  0.545  10   10
2      W   1  41  22   33  0.667  14   23  0.609   8   10  0.800   9   12
3      W   1  42  22   50  0.440  16   29  0.552   6   21  0.286  10   12
4      W   1  46  19   30  0.633  14   16  0.875   5   14  0.357  17   18

     FT%  ORB  DRB  TRB  AST  STL  BLK  TOV  PF  PTS  GmSc  Market_effect
0  0.923    2    6    8    6    3    1    5   3   70  54.5       1.555556
1  1.000    1   12   13    0    0    0    0   1   62  50.6       1.589744
2  0.750    3    4    7    4    0    0    2   2   61  48.8       1.487805
3  0.833    0    4    4    4    1    1    2   1   60  36.3       1.428571
4  0.944    2    8   10   11    4    1    5   2   60  56.6       1.304348
```

[121]:
```
# sum null values in each column
df.isna().sum()
```

[121]:
```
Rk                0
Player            0
Age               0
Pos               0
Date              0
Tm                0
Location          0
Opp               0
Result            0
GS                0
MP                0
FG                0
FGA               0
FG%            6023
2P                0
2PA               0
2P%           11092
3P                0
3PA               0
3P%           43504
FT                0
FTA               0
FT%           55577
ORB               0
DRB               0
TRB               0
AST               0
STL               0
```

```
BLK                0
TOV                0
PF                 0
PTS                0
GmSc               0
Maket_effect       0
dtype: int64
```

NaN values on DF refer to the percentage of shots made, if attemps == 0 then % is NaN,
If attemps >0 and no shots were good then % is 0

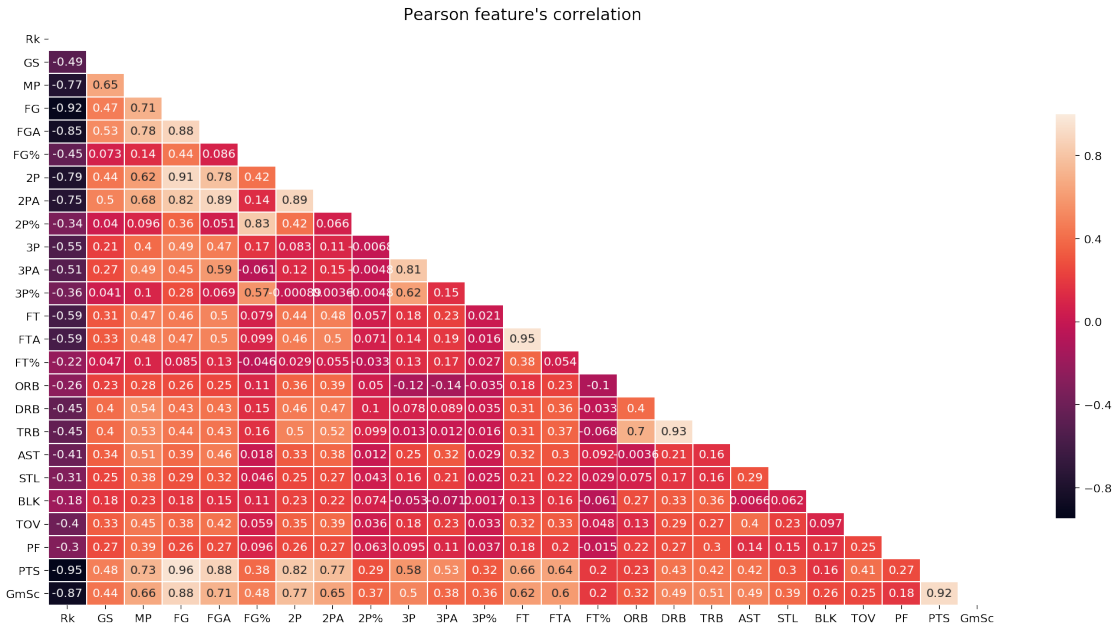
1.2 Features correlation

[12]:
```
df_corr = df.corr() #Pearson corr
```

[14]:
```
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(df_corr, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(18, 9))
plt.title("Pearson feature's correlation",fontsize=14)
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(df_corr, mask=mask, linewidths=.5, cbar_kws={"shrink": .
 ↪7},annot=True)


plt.savefig('features_corr_pearson.png')
plt.show()
```

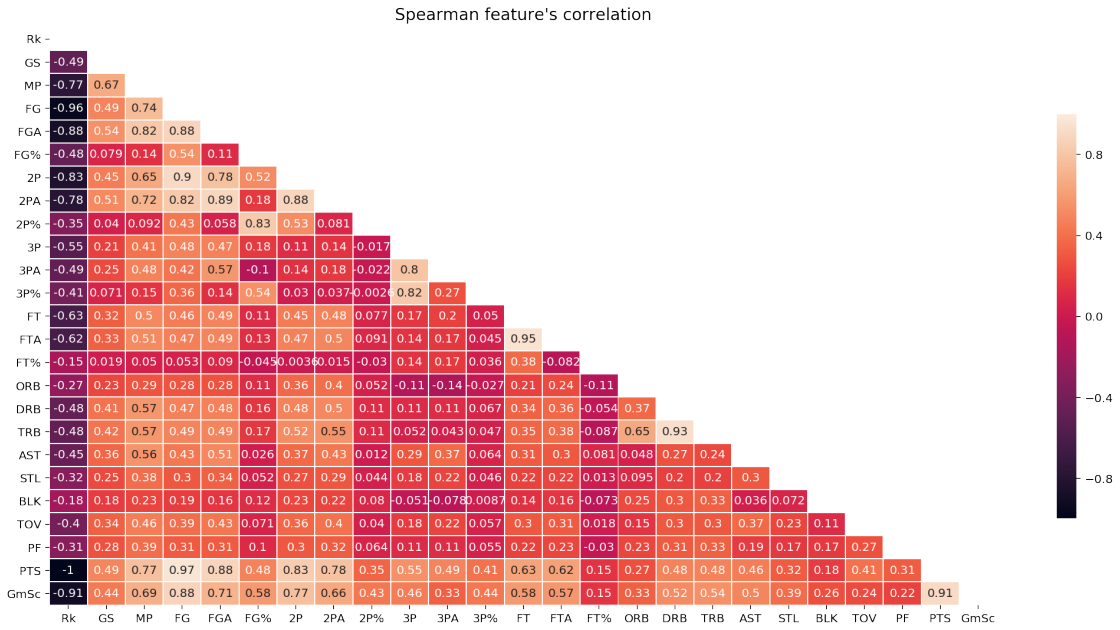


[15]:
```
df_corr_Spear = df.corr(method='spearman') #Spearman correlation
```

[16]:
```
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(df_corr_Spear, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(18, 9))
plt.title("Spearman feature's correlation",fontsize=14)
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(df_corr_Spear, mask=mask, linewidths=.5, cbar_kws={"shrink": .
 ↪7},annot=True)

plt.savefig('features_corr_spearman.png')
plt.show()
```



1.2.1 The correlation between minutes played 'MP' and points is:

1.2.2 Pearson = 0.73

1.2.3 Spearman = 0.77

1.2.4 Other feature might be interesting to check is if the player started the game 'GS'

1.2.5 Pearson = 0.48

1.2.6 Spearman = 0.49

2 XGboost regression model

[4]:
```
import xgboost as xgb
import graphviz
from sklearn.model_selection import cross_val_score, KFold,train_test_split
from sklearn.metrics import recall_score, precision_score, f1_score
```

[5]:
```
df = pd.read_csv('df_Basketball.csv')
pd.set_option('display.max_columns', None)
```

[148]:
```
df.head()
```

[148]:

	Rk	Player	Age	Pos	Date	Tm	Location	Opp	Result	GS	MP	FG	FGA	FG%	2P	2PA	2P%	3P	3PA	3P%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	GmSc	Market_effect
0	1	Devin Booker\bookede01	20-145	G	2017-03-24	PHO	@	BOS	L	1	45	21	40	0.525	17	29	0.586	4	11	0.364	24	26	0.923	2	6	8	6	3	1	5	3	70	54.5	1.555556
1	2	Carmelo Anthony\anthoca01	29-240	F	2014-01-24	NYK	vs	CHA	W	1	39	23	35	0.657	17	24	0.708	6	11	0.545	10	10	1.000	1	12	13	0	0	0	0	1	62	50.6	1.589744
2	3	LeBron James\jamesle01	29-063	F-G	2014-03-03	MIA	vs	CHA	W	1	41	22	33	0.667	14	23	0.609	8	10	0.800	9	12	0.750	3	4	7	4	0	0	2	2	61	48.8	1.487805
3	4	Kobe Bryant\bryanko01	37-234	G-F	2016-04-13	LAL	vs	UTA	W	1	42	22	50	0.440	16	29	0.552	6	21	0.286	10	12	0.833	0	4	4	4	1	1	2	1	60	36.3	1.428571
4	5	James Harden\hardeja01	28-157	G	2018-01-30	HOU	vs	ORL	W	1	46	19	30	0.633	14	16	0.875	5	14	0.357	17	18	0.944	2	8	10	11	4	1	5	2	60	56.6	1.304348

2.1 Minutes played as feature for points prediction

[7]:
```
# training data, feature to_frame returns a DF version of the values for␣
 ↪XGBoost
X = df.MP.to_frame()

# target variable
y = df.PTS

# create a Data matrix for cross validation in XGBoost
data_dmatrix = xgb.DMatrix(data=X,label=y)

params = {"objective":"reg:squarederror",'learning_rate': 0.2,'alpha': 10}
#                 'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=5,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",␣
 ↪as_pandas=True, seed=24)
```

[8]:
```
cv_results.tail()
```

[8]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
24	5.335778	0.003558	5.339713	0.014263
25	5.335697	0.003561	5.339650	0.014252
26	5.335643	0.003576	5.339610	0.014228
27	5.335612	0.003560	5.339588	0.014222
28	5.335577	0.003561	5.339578	0.014231

[9]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
28    5.339578
Name: test-rmse-mean, dtype: float64
```

2.2 Game started as feature for points prediction

[10]:
```
# training data, feature to_frame returns a DF version of the values for␣
 ↪XGBoost
X = df.GS.to_frame()

# target variable
y = df.PTS

# create a Data matrix for cross validation in XGBoost
data_dmatrix = xgb.DMatrix(data=X,label=y)

params = {"objective":"reg:squarederror",'learning_rate': 0.2,'alpha': 10}
#                 'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=5,
```

```
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",␣
 ↪as_pandas=True, seed=24)
```

[11]:
```
cv_results.tail()
```

[11]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
31	6.938075	0.004210	6.938184	0.016969
32	6.938034	0.004207	6.938209	0.016944
33	6.938111	0.004284	6.938182	0.016958
34	6.938074	0.004263	6.938183	0.016959
35	6.938084	0.004263	6.938180	0.016962

[12]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
35    6.93818
Name: test-rmse-mean, dtype: float64
```

2.3 Both features for points predictions

[13]:
```
# training data, feature to_frame returns a DF version of the values for␣
 ↪XGBoost
X = df[['MP','GS']]

# target variable
y = df.PTS

# create a Data matrix for cross validation in XGBoost
data_dmatrix = xgb.DMatrix(data=X,label=y)

params = {"objective":"reg:squarederror",'learning_rate': 0.2,'alpha': 10}
#                 'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=5,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",␣
 ↪as_pandas=True, seed=24)
```

[14]:
```
cv_results.tail()
```

[14]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
23	5.329099	0.003562	5.334717	0.014563
24	5.328965	0.003565	5.334611	0.014532
25	5.328879	0.003570	5.334564	0.014530
26	5.328815	0.003560	5.334535	0.014508
27	5.328792	0.003564	5.334519	0.014491

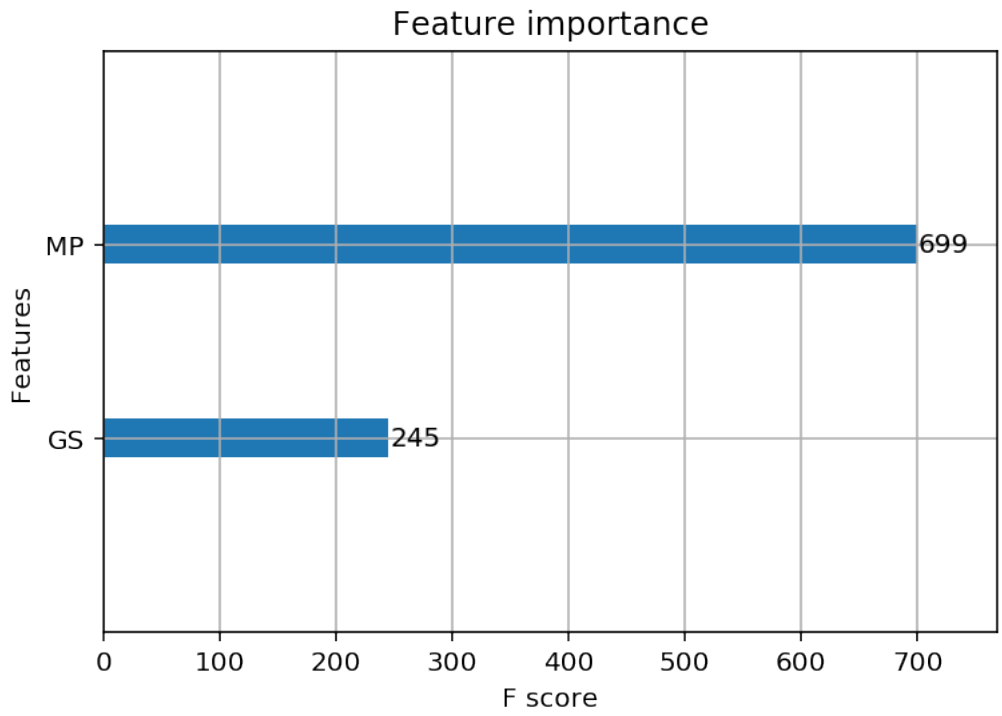
[15]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
27    5.334519
Name: test-rmse-mean, dtype: float64
```

[16]:
```
xg_reg = xgb.train(params=params, dtrain=data_dmatrix, num_boost_round=30)
```

[17]:
```
xgb.plot_importance(xg_reg)

plt.savefig('feature_importance_PTS.png')
plt.show()
```



3 Model training using MP as feature, and params from cv models

[18]:
```
# training data, feature to_frame returns a DF version of the values for␣
 ↪XGBoost
X = df.MP.to_frame()

# target variable
y = df.PTS

# create a Data matrix for cross validation in XGBoost
data_dmatrix = xgb.DMatrix(data=X,label=y)

#optimal params found with previous xgb.cv model

params = {"objective":"reg:squarederror",'learning_rate': 0.2,'alpha': 10}
num_boost_round=28
```

```
#model
xg_reg = xgb.train(params=params, dtrain=data_dmatrix,␣
 ↪num_boost_round=num_boost_round)
```

[19]:
```
import joblib
```

[22]:
```
#save model
joblib.dump(xg_reg,'xgb_pts_model.pkl' )

#load saved model
# xg_reg = joblib.load('xgb_pts_model.pkl')
```

[22]:
```
['xgb_pts_model.pkl']
```

[23]:
```
type(xg_reg)
```

[23]:
```
xgboost.core.Booster
```

Minutes played can be used to predict the total points a given player will score. One way to obtain the numer of minutes they will play, since we don't actually have that number, is to obtain the mean of the previous N games they have played. We could also add to the model if the player will be starting a game or not.

First, need to filter the players involved in a given match, probably only the ones involved in the most recent years

---

[24]:
```
# sort the data by Date
df_sort_date = df.sort_values(by=['Date'],ascending=False)

#reseting the index of df
df_sort_date = df_sort_date.reset_index(drop=True)
```

3.1 Teams

[25]:
```
teams_set = set(df_sort_date.Tm)
print(f'There are {len(teams_set)} teams in data frame.')
```

```
There are 31 teams in data frame.
```

3.2 Filtering for a match

[26]:
```
#Filtering the df for playing teams
tm1 = 'NYK'
tm2 = 'CHI'
```

```
if (tm1 not in teams_set) | (tm2 not in teams_set):
    print("Team not found")
else:
    print("Teams found in data")
```

```
Teams found in data
```

[27]:
```
#filter the data frame with teams conditional, result is a DF just for the␣
 ↪match between tm1 and tm2
game_df = df_sort_date[((df_sort_date['Tm'] == tm1) & (df_sort_date['Opp'] ==␣
 ↪tm2)) |((df_sort_date['Tm'] == tm2) & (df_sort_date['Opp'] == tm1))]
```

[28]:
```
# change column type to date time
game_df.Date = pd.to_datetime(game_df.Date)
```

```
/Users/ciro/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py:5303:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

[29]:
```
# create a column with Year
game_df['Year'] = game_df.Date.dt.year
```

```
/Users/ciro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[30]:
```
# the most recent year the match has been played
recent_year = game_df.Date.iloc[0].year
recent_year
```

[30]:
```
2018
```

[31]:
```
# new DF containing only the matches played in the most recent year
recent_game_df = game_df[game_df['Year'] == recent_year]
```

[32]:
```
recent_game_df.shape
```

[32]:
```
(42, 34)
```

3.3 Players

[33]:
```
# the list of players is created from the DF containing just the matches played␣
 ↪in the most recent year
# to avoid getting players from other seasons
players_list = list(set(recent_game_df.Player))
print(f'There are {len(players_list)} different players in data.')
```

```
There are 30 different players in data.
```

[149]:
```
# testing to obtain the mean of MP of last 10 games for each player

# pl1 = players_list[25]

# MP_mean_10g = df_sort_date[df_sort_date.Player == pl1].iloc[0:10].MP.mean()
# MP_mean_10g
```

[35]:
```
# creating a list with all the players involved in the last year games and␣
 ↪their corresponding mean of Minutes Played on their
# last 10 matches
player_MP_list = []
for pl in players_list:
    # using the DF sorted by date to obtain the most recent played matches of␣
 ↪every player
    mean10 = df_sort_date[df_sort_date.Player == pl].iloc[0:10].MP.mean()
    player_MP_list.append([pl,mean10])
```

[38]:
```
player_MP_list
```

[38]:
```
[['Trey Burke\\burketr01', 33.4],
 ['Lauri Markkanen\\markkla01', 25.0],
 ['Bobby Portis\\portibo01', 23.1],
 ['Luke Kornet\\kornelu01', 20.7],
 ['Ron Baker\\bakerro01', 10.5],
 ['Denzel Valentine\\valende01', 25.4],
 ['Emmanuel Mudiay\\mudiaem01', 19.0],
 ['Doug McDermott\\mcderdo01', 19.8],
 ['Cristiano Felício\\feliccr01', 25.8],
 ['Troy Williams\\willitr02', 18.6],
 ['Noah Vonleh\\vonleno01', 23.6],
 ['Tim Hardaway\\hardati02', 33.2],
 ['Kristaps Porziis\\porzikr01', 29.0],
 ['Lance Thomas\\thomala01', 19.5],
 ['Enes Kanter\\kanteen01', 25.5],
 ['Jarrett Jack\\jackja01', 16.6],
 ['Damyean Dotson\\dotsoda01', 18.2],
 ["Kyle O'Quinn\\oquinky01", 22.9],
 ['Isaiah Hicks\\hicksis01', 14.0],
 ['Antonio Blakeney\\blakean01', 16.8],
```

```
['David Nwaba\\nwabada01', 28.0],
['Cameron Payne\\payneca01', 25.7],
['Michael Beasley\\beaslmi01', 26.3],
['Courtney Lee\\leeco01', 25.8],
['Jerian Grant\\grantje02', 24.1],
['Frank Ntilikina\\ntilila01', 28.7],
['Paul Zipser\\zipsepa01', 15.5],
['Robin Lopez\\lopezro01', 19.0],
['Kris Dunn\\dunnkr01', 29.0],
['Justin Holiday\\holidju01', 23.4]]
```

[71]:
```
# create an array of MP for each player involved in the match
X_to_pred = np.array([x[1] for x in player_MP_list])
```

[93]:
```
dataset = pd.DataFrame()
dataset['Player'] = np.array([x[0] for x in player_MP_list])
dataset['MP']=X_to_pred
```

[94]:
```
dataset.tail()
# X_to_pred
```

[94]:

	Player	MP
25	Frank Ntilikina\ntilila01	28.7
26	Paul Zipser\zipsepa01	15.5
27	Robin Lopez\lopezro01	19.0
28	Kris Dunn\dunnkr01	29.0
29	Justin Holiday\holidju01	23.4

[96]:
```
# create a Data matrix for XGBoost model
data_dmatrix = xgb.DMatrix(data=dataset.MP.to_frame())
```

[100]:
```
# predicted pts of each player using xg_reg model
pts_pred = xg_reg.predict(data_dmatrix)
dataset['PTS_pred']= pts_pred
```

[102]:
```
dataset.PTS_pred = round(dataset.PTS_pred,2)
```

[103]:
```
dataset
```

[103]:

	Player	MP	PTS_pred
0	Trey Burke\burketr01	33.4	15.30
1	Lauri Markkanen\markkla01	25.0	10.10
2	Bobby Portis\portibo01	23.1	8.79
3	Luke Kornet\kornelu01	20.7	7.86
4	Ron Baker\bakerro01	10.5	3.23
5	Denzel Valentine\valende01	25.4	10.10
6	Emmanuel Mudiay\mudiaem01	19.0	6.72
7	Doug McDermott\mcderdo01	19.8	7.24
8	Cristiano Felício\feliccr01	25.8	10.66
9	Troy Williams\willitr02	18.6	6.72
10	Noah Vonleh\vonleno01	23.6	9.44

```
11        Tim Hardaway\hardati02  33.2     15.30
12  Kristaps Porziis\porzikr01  29.0     12.51
13        Lance Thomas\thomala01  19.5      7.24
14         Enes Kanter\kanteen01  25.5     10.66
15         Jarrett Jack\jackja01  16.6      5.77
16      Damyean Dotson\dotsoda01  18.2      6.33
17        Kyle O'Quinn\oquinky01  22.9      8.79
18        Isaiah Hicks\hicksis01  14.0      4.58
19    Antonio Blakeney\blakean01  16.8      5.77
20         David Nwaba\nwabada01  28.0     12.00
21       Cameron Payne\payneca01  25.7     10.66
22     Michael Beasley\beaslmi01  26.3     10.66
23          Courtney Lee\leeco01  25.8     10.66
24        Jerian Grant\grantje02  24.1      9.44
25     Frank Ntilikina\ntilila01  28.7     12.51
26         Paul Zipser\zipsepa01  15.5      5.37
27         Robin Lopez\lopezro01  19.0      6.72
28            Kris Dunn\dunnkr01  29.0     12.51
29      Justin Holiday\holidju01  23.4      8.79
```

4 Top 10 players per market

[145]:
```
market = 'PTS'
df_effectv_sorted = df.sort_values(by=[market],ascending=False)
```

[153]:
```
df_effectv_sorted.head(10)
```

[153]:

	Rk	Player	Age	Pos	Date	Tm	Location	Opp	Result	GS	MP	FG	FGA	FG%	2P	2PA	2P%	3P	3PA	3P%	FT
0	1	Devin Booker\bookede01	20-145	G	2017-03-24	PHO	@	BOS	L	1	45	21	40	0.525	17	29	0.586	4	11	0.364	24
1	2	Carmelo Anthony\anthoca01	29-240	F	2014-01-24	NYK	vs	CHA	W	1	39	23	35	0.657	17	24	0.708	6	11	0.545	10
2	3	LeBron James\jamesle01	29-063	F-G	2014-03-03	MIA	vs	CHA	W	1	41	22	33	0.667	14	23	0.609	8	10	0.800	9
3	4	Kobe Bryant\bryanko01	37-234	G-F	2016-04-13	LAL	vs	UTA	W	1	42	22	50	0.440	16	29	0.552	6	21	0.286	10
4	5	James Harden\hardeja01	28-157	G	2018-01-30	HOU	vs	ORL	W	1	46	19	30	0.633	14	16	0.875	5	14	0.357	17
5	6	Klay Thompson\thompkl01	26-301	G-F	2016-12-05	GSW	vs	IND	W	1	29	21	33	0.636	13	19	0.684	8	14	0.571	10
6	7	Anthony Davis\davisan02	22-347	F-C	2016-02-21	NOP	@	DET	W	1	43	24	34	0.706	22	32	0.688	2	2	1.000	9
7	8	Damian Lillard\lillada01	26-267	G	2017-04-08	POR	vs														
8	9	Russell Westbrook\westbru01	28-115	G	2017-03-07	OKC	vs														
9	10	Kyrie Irving\irvinky01	22-354	G	2015-03-12	CLE	@														

7	UTA	W	1	42	18	34	0.529	9	20	0.450	9	14	0.643	14
8	POR	L	1	36	21	39	0.538	18	30	0.600	3	9	0.333	13
9	SAS	W	1	47	20	32	0.625	13	25	0.520	7	7	1.000	10

	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	GmSc
0	26	0.923	2	6	8	6	3	1	5	3	70	54.5
1	10	1.000	1	12	13	0	0	0	0	1	62	50.6
2	12	0.750	3	4	7	4	0	0	2	2	61	48.8
3	12	0.833	0	4	4	4	1	1	2	1	60	36.3
4	18	0.944	2	8	10	11	4	1	5	2	60	56.6
5	11	0.909	0	2	2	1	0	0	0	0	60	46.2
6	10	0.900	6	14	20	4	0	1	2	1	59	53.9
7	16	0.875	0	6	6	5	1	1	0	0	59	48.6
8	16	0.813	1	2	3	9	3	1	4	4	58	43.6
9	10	1.000	1	2	3	5	4	0	2	3	57	48.2

[ ]: