# Basketball_Markets

September 9, 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')
```

[67]:
```
# df.head()
```

[66]:
```
# sum null values in each column
# df.isna().sum()
```

### 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

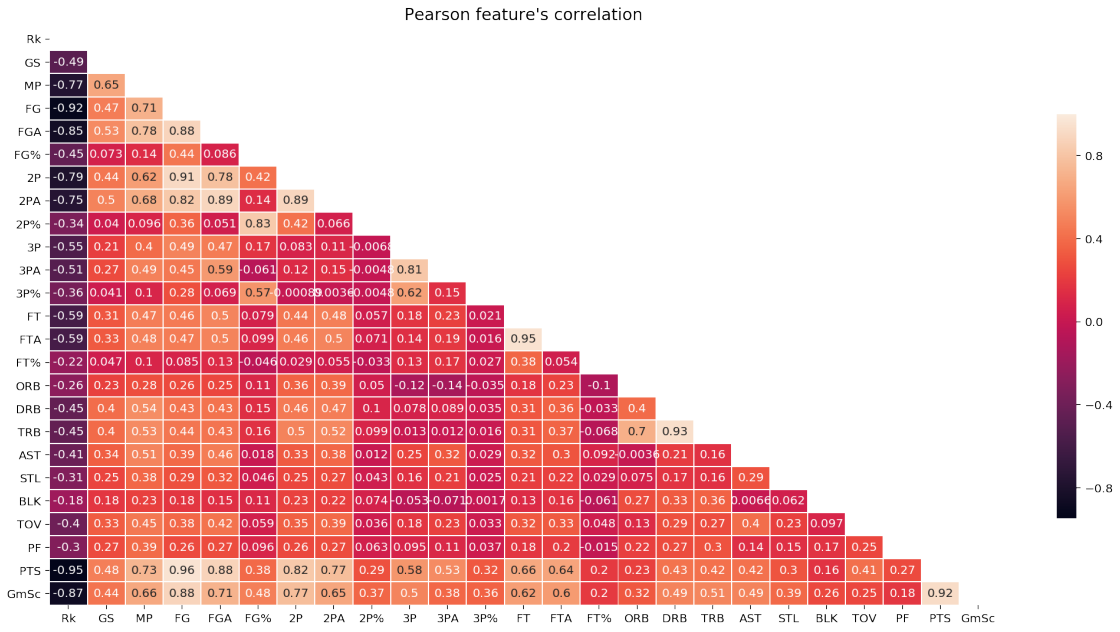
## 1.2 Features correlation

[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()
```



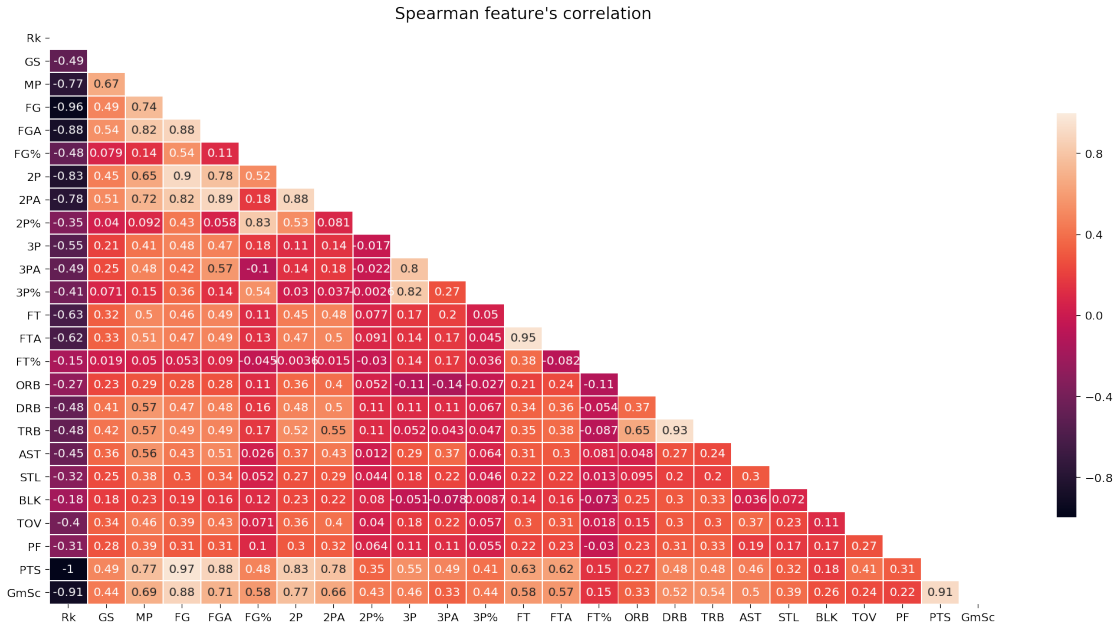

[15]:
```
df_corr_Spear = df.corr(method='spearman') #Spearman correlation
```

[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)
```

### 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 |

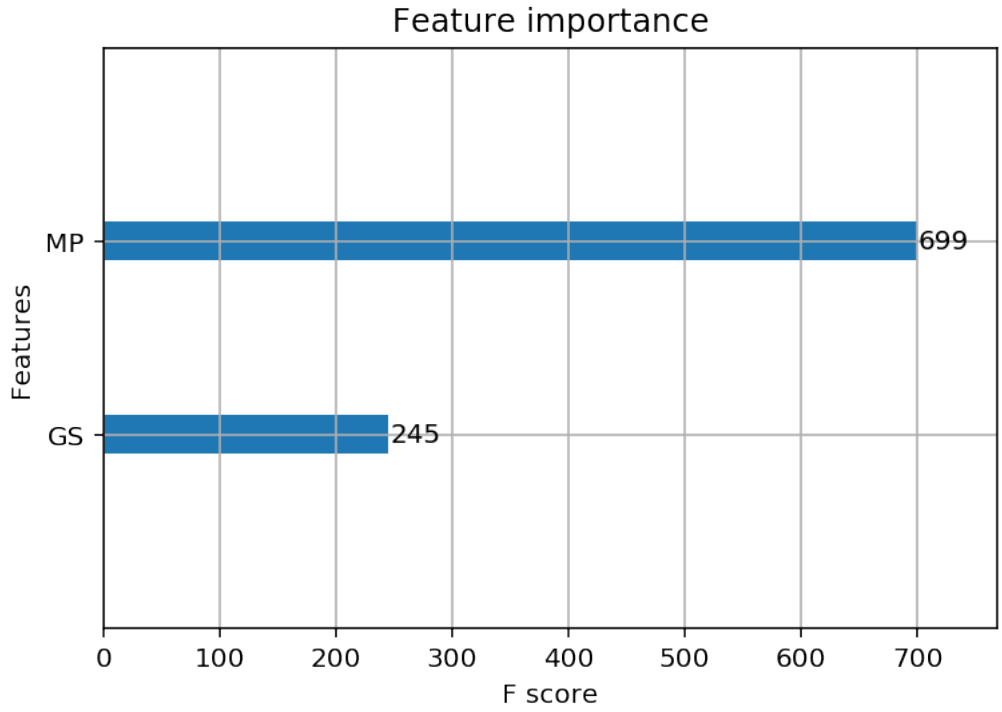
[15]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
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)
```

[50]:
```
import joblib
```

[51]:
```
#save model
# joblib.dump(xg_reg,'xgb_pts_model.pkl' )

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

[52]:
```
type(xg_reg)
```

[52]:
```
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

---

[7]:
```
# 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

[8]:
```
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

[9]:
```
#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
```

[10]:
```
#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))]
```

[11]:
```
# 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
```

[12]:
```
# 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
```

[13]:
```
# the most recent year the match has been played
recent_year = game_df.Date.iloc[0].year
recent_year
```

[13]: 2018

[14]:
```
# new DF containing only the matches played in the most recent year
recent_game_df = game_df[game_df['Year'] == recent_year]
```

[15]:
```
recent_game_df.shape
```

[15]:
```
(42, 34)
```

### 3.3 Players

[16]:
```
# 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.
```

[17]:
```
# 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
```

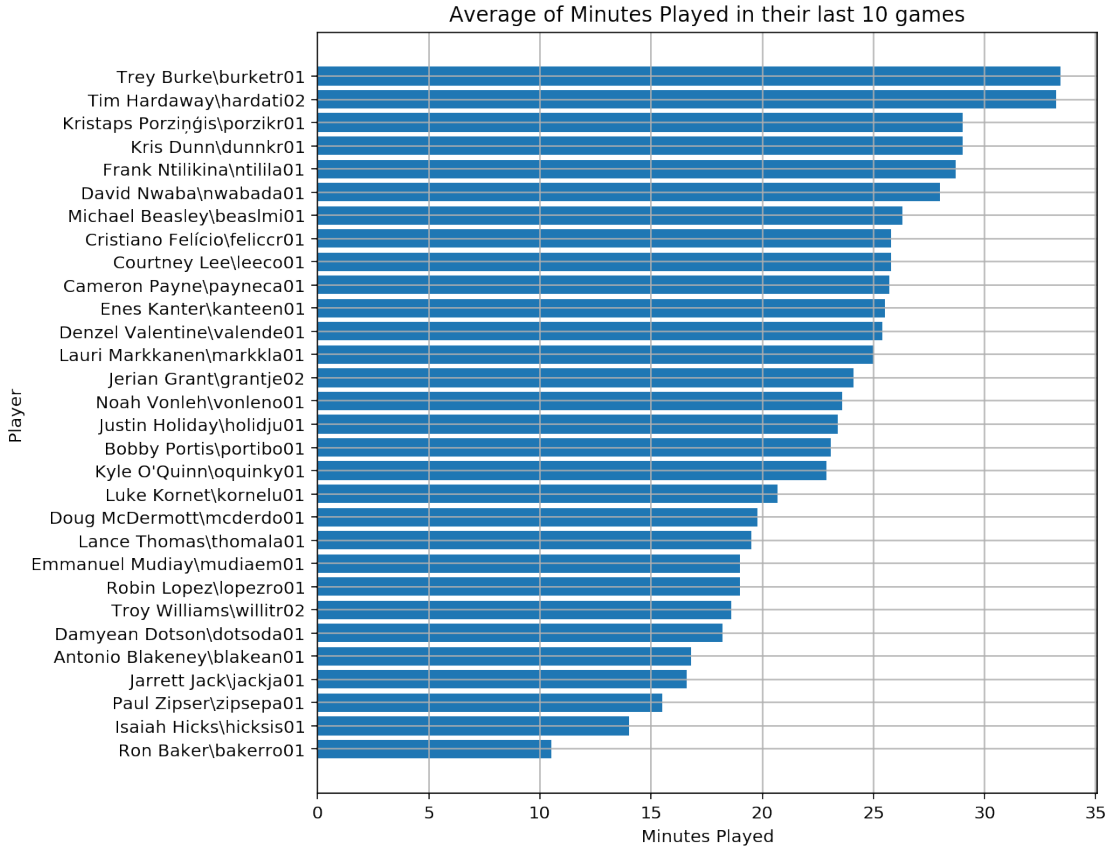
[18]:
```
# creating a list with all the players involved in the last year games and␣
 ↪their corresponding mean of Minutes Played on their
# last 10 matches
player_MP_list = []
for pl in players_list:
    # using the DF sorted by date to obtain the most recent played matches of␣
 ↪every player
    mean10 = df_sort_date[df_sort_date.Player == pl].iloc[0:10].MP.mean()
    player_MP_list.append([pl,mean10])
```

[41]:
```
#sort list of players by MP
sorted_list = sorted(player_MP_list, key=lambda x: x[1])
```

[42]:
```
names = [x[0] for x in sorted_list]
MP_played_mean = [x[1] for x in sorted_list]
```

[43]:
```
fig = plt.figure(figsize=[8,8])
ax = plt.barh(y=names,width=MP_played_mean,height=0.8)
plt.grid()
plt.title('Average of Minutes Played in their last 10 games')
plt.xlabel('Minutes Played')
plt.ylabel('Player')

plt.show()
```




[44]:
```
dataset = pd.DataFrame()
dataset['Player'] = names
dataset['MP']= MP_played_mean
```

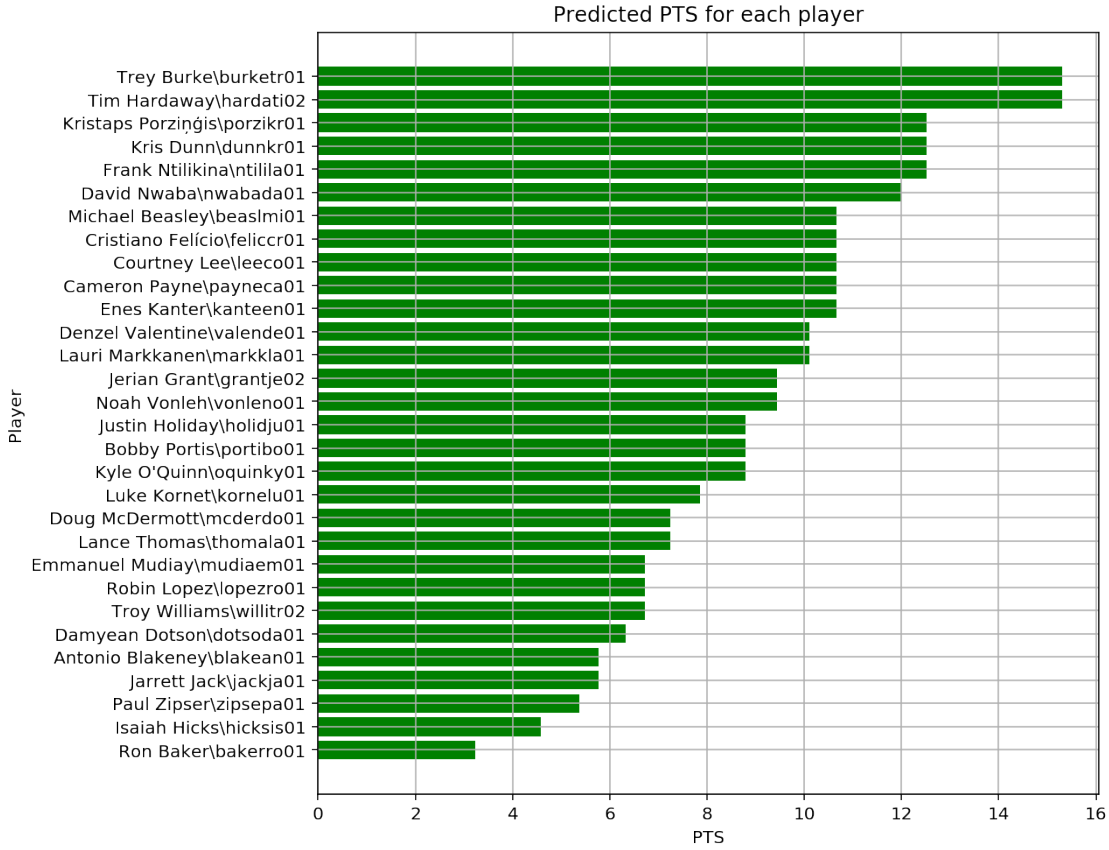
[47]:
```
# create a Data matrix for XGBoost model
data_dmatrix = xgb.DMatrix(data=dataset.MP.to_frame())
```

[53]:
```
# predicted pts of each player using xg_reg model
pts_pred = xg_reg.predict(data_dmatrix)
dataset['PTS_pred']= pts_pred
```

[54]:
```
dataset.PTS_pred = round(dataset.PTS_pred,2)
```

[64]:
```
fig = plt.figure(figsize=[8,8])
ax = plt.barh(y=dataset['Player'],width=dataset['PTS_pred'],height=0.
 ↪8,color='green')
plt.grid()
plt.title('Predicted PTS for each player')
plt.xlabel('PTS')
plt.ylabel('Player')

plt.show()
```

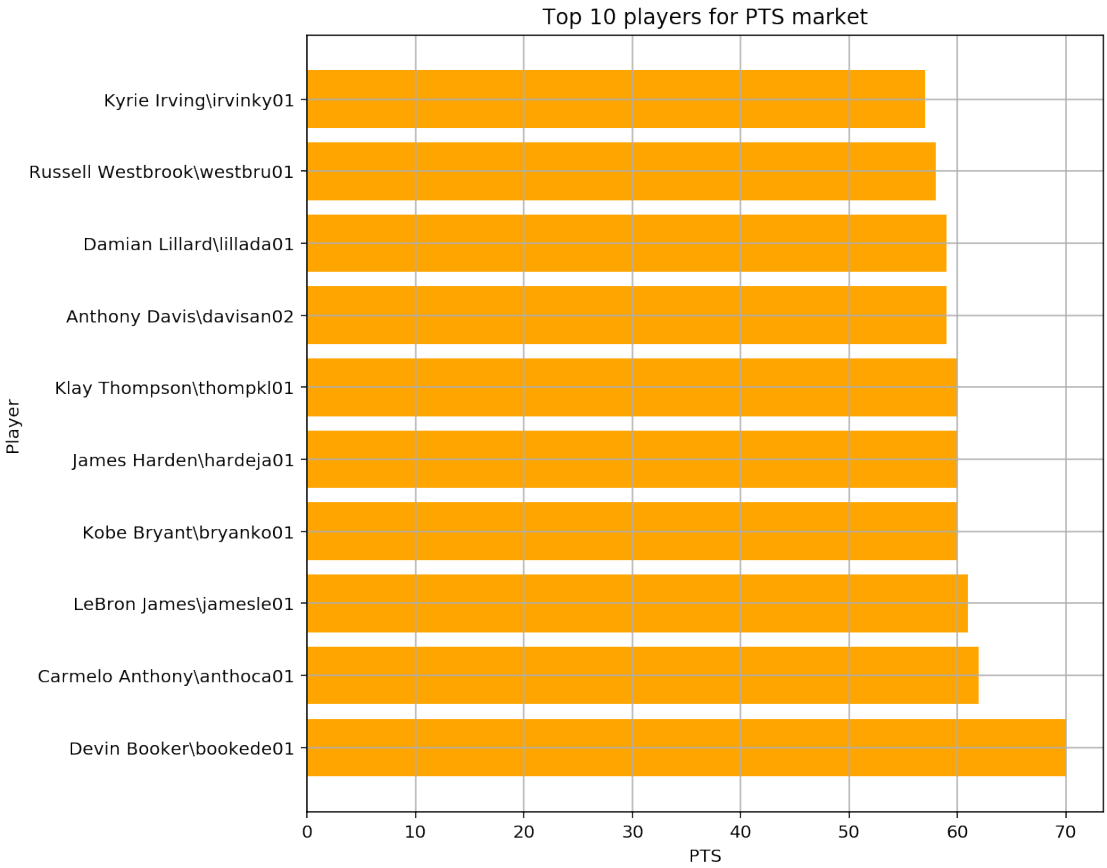



# 4 Top 10 players per market

[58]:
```
market = 'PTS'
df_effectv_sorted = df.sort_values(by=[market],ascending=False)
```

[65]:
```
fig = plt.figure(figsize=[8,8])
ax = plt.barh(y=df_effectv_sorted['Player'].iloc[0:
 ↪10],width=df_effectv_sorted[market].iloc[0:10],height=0.8,color='orange')
plt.grid()
plt.title(f'Top 10 players for {market} market')
plt.xlabel(f'{market}')
plt.ylabel('Player')

plt.show()
```




## 5 Other markets and Rookies

---

Other markets can also be predicted using the mean of minutes played by each player in their last 10 games, however, the correlation between markets such as rebounds, assists, steals and blocks and minutes played is not as strong as with total points.

Another approcah, maybe more complex, could be to do a time series to predict the number of minutes for a player in his next match.

The top 10 players in other markets can easily be plotted following the example above.

In terms of rookies, since we don't have any previous data on them, we could pick all the rookies from the last season in the data and get the average of minutes played for a rookie and use that value for prediction.

---

[ ]: