Using Python Pandas with NBA Data

Justin Jacobs

8 May 2016

Abstract

The goal of this document is to go through a series of basic commands using Python's Pandas functionality to analyze a sample NBA data file. The datafile is a comma separated value (csv) file that will be read as a Pandas dataframe.

1 Python: Data Manipulation

Python has a large amount of functionality that crosses between C-programming, MATLAB computing, and basic scripting data manipulation programs such as Perl and Unix command line. This hybrid functionality makes Python sexy to novice programmers and, combined with a large community on sites as Stack-Exchange, allows for effective processing of data without requiring to write large amounts of code to handle specialized data types.

In this document, we look into using the Pandas package for analyzing data within the Python environment. Pandas is attractive as it can store csv files as a **dataframe**; which is equivalent to viewing data as a Microsoft Excel spreadsheet. This functionality will allow us to perform simple calculations and index files for further processing using other packages available in Python.

To begin, we open a terminal and move to the directory for which our files are located. We can do this multiple ways by either just directory walking or scanning all files in the directory. Here, instead, we will focus solely on one file. The file of interest for this document is **[2016-03-12]-0021500979-OKC@SAS.csv**. This is the March 12th game between the Oklahoma City Thunder and the San Antonio Spurs in San Antonio, Texas.

In the directory of preference, we open Python by merely typing:

```
user$ python
```

This will open a Python prompt:

```
Python 2.7.10 (default, Oct 23 2015, 18:05:06)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Here, you see we are operating with Python version 2.7.10. Using version 3.x.x is certainly acceptable and maybe even preferred. This just shows, we can do some archaic work in version 2.x.x. Next, we set the file name we wish to operate on. We only set this string variable in case we do a future directory walk and we swap out file names into the data ingest function.

```
>>> file = '[2016-03-12]-0021500979-OKC@SAS.csv'
>>> file
'[2016-03-12]-0021500979-OKC@SAS.csv'
```

2 Pandas

Now that the data file is ready to be ingested into a Pandas dataframe, we must import Pandas. Normally, Pandas does not come included with Python. Instead, it must be downloaded and installed. The easiest method to do this is to perform a **pip install pandas** command back on the command line. Clear instructions on setting up pip for install and using pip to install pandas can be found on the Stack open source pages. Once it is installed, we call

```
>>> import pandas as pd
```

This imports the Pandas package and labels it as **pd**. Under this labeling, any functions called from the Pandas class will be **pd.functionName**. Hence the **pd** is just shorthand for Pandas. We could, theoretically, set it to anything we want. Here, **pd** is common and suffices.

2.1 Reading In Files and Accessing Data

The first function we care about is the file read function. Since our data is set up as a csv file, we can use the **readcsv** function. The csv file has 476 rows across 44 columns. The values in the columns will all be strings; despite being either numerical or string valued. First, we ingest the csv file into a dataframe:

```
>>> game = pd.read_csv(file)
```

We set the dataframe to be called **game**. We can check the shape of the dataframe by using the command:

```
>>> game.shape
(476, 44)
```

To which we see that there are indeed 476 rows and 44 columns. We can view the column names by using:

```
>>> list(game.columns.values)
['game_id', 'data_set', 'date', 'a1', 'a2', 'a3', 'a4', 'a5', 'h1',
 'h2', 'h3', 'h4', 'h5', 'period', 'away_score', 'home_score',
 'remaining_time', 'elapsed', 'play_length', 'play_id', 'team',
 'event_type', 'assist', 'away', 'home', 'block', 'entered', 'left',
 'num', 'opponent', 'outof', 'player', 'points', 'possession',
 'reason', 'result', 'steal', 'type', 'shot_distance', 'original_x',
 'original_y', 'converted_x', 'converted_y', 'description']
```

The attribute **columns** gives a reference index list. The sub-attribute **values** removes the string characters and index characters to produce an array of string values; the headers wrapped in an array data struct. point this to a **list** removes the string array struct notation and produces a list of column headers in order of which they are read into the dataframe.

We can call usual **head** and **tail** commands:

```
>>> game.head(5)
         game_id                  data_set        date            a1  \
0  ="0021500979"  2015-2016 Regular Season  2016-03-12  Kevin Durant
1  ="0021500979"  2015-2016 Regular Season  2016-03-12  Kevin Durant
2  ="0021500979"  2015-2016 Regular Season  2016-03-12  Kevin Durant
3  ="0021500979"  2015-2016 Regular Season  2016-03-12  Kevin Durant
4  ="0021500979"  2015-2016 Regular Season  2016-03-12  Kevin Durant

            a2            a3              a4                 a5  \
0  Serge Ibaka  Steven Adams  Andre Roberson  Russell Westbrook
1  Serge Ibaka  Steven Adams  Andre Roberson  Russell Westbrook
2  Serge Ibaka  Steven Adams  Andre Roberson  Russell Westbrook
3  Serge Ibaka  Steven Adams  Andre Roberson  Russell Westbrook
4  Serge Ibaka  Steven Adams  Andre Roberson  Russell Westbrook

              h1                 h2  \
0  Kawhi Leonard  LaMarcus Aldridge
1  Kawhi Leonard  LaMarcus Aldridge
2  Kawhi Leonard  LaMarcus Aldridge
3  Kawhi Leonard  LaMarcus Aldridge
4  Kawhi Leonard  LaMarcus Aldridge

                         ...                            reason  result  \
0                        ...                               NaN     NaN
1                        ...                               NaN     NaN
2                        ...                         lost ball     NaN
3                        ...                               NaN  missed
4                        ...                               NaN     NaN
```

```
        steal               type  shot_distance  original_x original_y  \
0         NaN    start of period            NaN         NaN        NaN
1         NaN          jump ball            NaN         NaN        NaN
2  Tim Duncan         lost ball            NaN         NaN        NaN
3         NaN          Jump Shot             25         119        223
4         NaN  rebound defensive            NaN         NaN        NaN

  converted_x converted_y                                        description
0         NaN         NaN                                                NaN
1         NaN         NaN       Jump Ball Duncan vs. Adams: Tip to Westbrook
2         NaN         NaN  Westbrook Lost Ball Turnover (P1.T1), Duncan S...
3        36.9        66.7                      MISS Leonard 25' 3PT Jump Shot
4         NaN         NaN                      Roberson REBOUND (Off:0 Def:1)

[5 rows x 44 columns]


>>> game.tail(5)
           game_id                  data_set        date                 a1  \
471  ="0021500979"  2015-2016 Regular Season  2016-03-12  Russell Westbrook
472  ="0021500979"  2015-2016 Regular Season  2016-03-12  Russell Westbrook
473  ="0021500979"  2015-2016 Regular Season  2016-03-12  Russell Westbrook
474  ="0021500979"  2015-2016 Regular Season  2016-03-12  Russell Westbrook
475  ="0021500979"  2015-2016 Regular Season  2016-03-12  Russell Westbrook

              a2            a3              a4              a5             h1  \
471  Serge Ibaka  Kevin Durant  Andre Roberson  Anthony Morrow  Kawhi Leonard
472  Serge Ibaka  Kevin Durant  Anthony Morrow      Randy Foye  Kawhi Leonard
473  Serge Ibaka  Kevin Durant  Anthony Morrow      Randy Foye  Kawhi Leonard
474  Serge Ibaka  Kevin Durant  Anthony Morrow      Randy Foye  Kawhi Leonard
475  Serge Ibaka  Kevin Durant  Anthony Morrow      Randy Foye  Kawhi Leonard

              h2                    ...                    reason  result  \
471  Danny Green                    ...                       NaN     NaN
472  Danny Green                    ...                       NaN     NaN
473  Danny Green                    ...                       NaN  missed
474  Danny Green                    ...                       NaN     NaN
475  Danny Green                    ...                       NaN     NaN

    steal                       type  shot_distance  original_x original_y  \
471   NaN                        sub            NaN         NaN        NaN
472   NaN                        sub            NaN         NaN        NaN
473   NaN  Driving Finger Roll Layup              1          -1          8
474   NaN          rebound defensive            NaN         NaN        NaN
475   NaN              end of period            NaN         NaN        NaN
```

```
    converted_x converted_y                              description
471         NaN         NaN                      SUB: Morrow FOR Adams
472         NaN         NaN                     SUB: Foye FOR Roberson
473        25.1         5.8  MISS Morrow 1' Driving Finger Roll Layup
474         NaN         NaN            Aldridge REBOUND (Off:2 Def:7)
475         NaN         NaN                                       NaN

[5 rows x 44 columns]
```

If we wish to access a particular column of data, we call the data by its labels. For instance, suppose we extract the away score for every NBA action in the file. The column name is **away_score**. This will produce a left-indexed pandas column array.

```
>>> game['away_score']
0       0
1       0
2       0
3       0
4       0
5       0
6       0
7       0
8       0
9       2
10      2
11      2
```

We can now reference this array easily by usual array calls:

```
>>> awayScore = game['away_score']
>>> awayScore[10]
2
```

To access row data, we turn to the **loc** command. This command will break out a Pandas object variable that indexes the columns across the row. Suppose we want the 11th action in the game. Since indices start at zero, we select index 10. The resulting object is a Pandas array indexed by the column headers.

```
>>> game.loc[10]
game_id                              ="0021500979"
data_set                  2015-2016 Regular Season
date                                   2016-03-12
a1                                    Kevin Durant
a2                                     Serge Ibaka
a3                                    Steven Adams
a4                                  Andre Roberson
```

```
a5                                  Russell Westbrook
h1                                      Kawhi Leonard
h2                                  LaMarcus Aldridge
h3                                         Tim Duncan
h4                                        Danny Green
h5                                        Tony Parker
period                                              1
away_score                                          2
home_score                                          0
remaining_time                               00:10:51
elapsed                                      00:01:09
play_length                                  00:00:22
play_id                                            10
team                                              SAS
event_type                                       miss
assist                                            NaN
away                                              NaN
home                                              NaN
block                                             NaN
entered                                           NaN
left                                              NaN
num                                               NaN
opponent                                          NaN
outof                                             NaN
player                              LaMarcus Aldridge
points                                              0
possession                                        NaN
reason                                            NaN
result                                         missed
steal                                             NaN
type                                Pullup Jump Shot
shot_distance                                      16
original_x                                       -119
original_y                                        100
converted_x                                      13.1
converted_y                                       79
description       MISS Aldridge 16' Pullup Jump Shot
Name: 10, dtype: object
```

Since the array is indexed by the column headers, we select an element of the row by calling the column name. For instance, if we select the first player for the away team, we do not select **[3]**, but rather **'a1'**.

```
>>> game.loc[10]['a1']
'Kevin Durant'
```

2.2 Querying Data in a Dataframe

Now that we know how to extract data from rows and columns, we can then start walking through the indices and start querying data. For instance, let's count how many actions that **Kevin Durant** has played in. To do this, we just walk through the Pandas dataframe and count the number of times Kevin Durant's name appears. By scanning the file manually, Durant's name appears in four of the five possible away team columns. Therefore we must scan across multiple columns as well as rows.

First we assume there is no assumption on the dataframe. This is the most basic assumption. That is, Durant's name can appear anywhere in the file. If we are not smart, we may count more instances than exists. This is due to the multiple columns where Durant's name can appear. An easy to perform this query is to walk through the rows and scan the columns until Durant's name is found. Once it is found, we break the column search.

```
>>> count = 0
>>> labels = list(game.columns.values)
>>> for i in range(game.shape[0]):
...     for label in labels:
...             if 'Kevin Durant' in str(game.loc[i][label]):
...                     count = count + 1
...                     print label
...                     break
...
>>> count
364
```

We initialize a counter called **count** to zero and walk through the rows of the dataframe, indicated by **game.shape[0]**. The column headers are stored in a string list called **labels**. As we iterate through the labels, we set the value of the row and column to a string so we can do a string comparison to **Kevin Durant**. If this is a match, we iterate the count, print the column header where we match and break the column search; iterating to the next row.

By printing the column header, we can identify if there are columns we do not anticipate to be counted. In fact, based on this code block, there indeed is. There is a series of actions where Durant is not listed on the floor, but rather is listed in the **left** column; meaning he is substituted out (or left the court) of the game. So let's eliminate these instances.

```
>>> count = 0
>>> for i in range(game.shape[0]):
...     if 'Kevin Durant' in str(game.loc[i]['a1':'h5']):
...             count = count + 1
...
>>> count
360
```

Here, we eliminated the need of the labels list since we know the structure of the dataframe. In this case, we select the ten players on the court, indicated by **'a1'** through **'h5'**. We did all ten players only because we want to recycle the code and count how many actions all players participated in. Here, we simplify this task by calling the **Counter** function contained in Python's **collections** package.

```
>>> from collections import Counter
>>> totals = Counter(list(game.loc[0,'a1':'h5']))
>>> totals
Counter({'Danny Green': 1, 'Tim Duncan': 1, 'Kevin Durant': 1,
'Tony Parker': 1, 'Serge Ibaka': 1, 'Russell Westbrook': 1,
'Andre Roberson': 1, 'LaMarcus Aldridge': 1, 'Steven Adams': 1,
'Kawhi Leonard': 1})


>>> for i in range(1,game.shape[0]):
...     totals = totals + Counter(list(game.loc[i,'a1':'h5']))
...
>>> totals
Counter({'Russell Westbrook': 375, 'Kawhi Leonard': 371,
'Serge Ibaka': 363, 'LaMarcus Aldridge': 363, 'Kevin Durant': 360,
'Andre Roberson': 315, 'Enes Kanter': 288, 'Danny Green': 275,
'Tony Parker': 275, 'David West': 268, 'Tim Duncan': 215,
'Kyle Singler': 208, 'Patty Mills': 201, 'Steven Adams': 199,
'Manu Ginobili': 177, 'Randy Foye': 175, 'Kyle Anderson': 104,
'Boris Diaw': 97, 'Nick Collison': 92, 'Kevin Martin': 34,
'Anthony Morrow': 5})
```

If we need to differentiate players between the teams, we can easily just split the code to scan **'a1'** through **'a5'** for the away team; analogous for the home team.

2.3 Grouping

If we wish to perform more sophisticated groupings, we can either expand out the queries into lists and string concatenation; or we can leverage the power of Pandas. We will follow the latter case and apply the **groupby** function in pandas. As a basic example, let's take a look at the five-on-five match-ups on the court. The **groupby** function creates a dictionary data structure with the key value being a the grouping across the columns of interest and the mapping being the row index.

```
>>> onCourt = game.groupby(['a1','a2','a3','a4','a5','h1','h2','h3','h4','h5'])
>>> onCourt.groups
{('Kyle Singler', 'Russell Westbrook', 'Serge Ibaka',
```

```
'Andre Roberson', 'Enes Kanter', 'Kyle Anderson', 'Tim Duncan',
'Tony Parker', 'Kawhi Leonard', 'LaMarcus Aldridge'): [168, 169, 170],
('Enes Kanter', 'Randy Foye', 'Russell Westbrook', 'Andre Roberson',
'Serge Ibaka', 'David West', 'Patty Mills', 'Boris Diaw', 'Kawhi Leonard',
'Danny Green'): [382], ('Kyle Singler', 'Randy Foye', 'Russell Westbrook',
'Serge Ibaka', 'Steven Adams', 'Boris Diaw', 'Kyle Anderson',
'Tim Duncan', 'Tony Parker', 'Kawhi Leonard'): [146, 147, 148, 149,
150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160], ...
```

The first dictionary element is **Kyle Singler, Russell Westbrook, Serge Ibaka, Andre Roberson, Enes Kanter, Kyle Anderson, Tim Duncan, Tony Parker, Kawhi Leonard, and LaMarcus Aldridge**. This pairing appears to occur for only action numbers **168, 169, 170**. This action was a substitution of Andre Roberson in for Steven Adams, a shooting foul by Kyle Singler on Kawhi Leonard (basket made) and a substitution of Kevin Durant in for Kyle Singler. However, the Python dictionary does not sort dictionary key-lists. So there may be more actions that involve the 10 players on the court.

As a quick example, we changed **'a1':'h5'** to **'a1':'a5'** and then print out the number of times the quintuple appeared together on the court. This is performed by merely counting values in any home player location; here we selected **'h4'**.

```
>>> onCourt['h4'].count()
```

a1	a2	a3	a4	a5	
Andre Roberson	Kyle Singler	Enes Kanter	Kevin Durant	Steven Adams	5
Enes Kanter	Kyle Singler	Randy Foye	Kevin Durant	Nick Collison	18
				Steven Adams	9
			Nick Collison	Russell Westbrook	1
	Randy Foye	Russell Westbrook	Andre Roberson	Serge Ibaka	9
	Russell Westbrook	Andre Roberson	Serge Ibaka	Kevin Durant	6
Kevin Durant	Enes Kanter	Kyle Singler	Randy Foye	Nick Collison	45
		Randy Foye	Nick Collison	Russell Westbrook	12
			Russell Westbrook	Andre Roberson	1
	Serge Ibaka	Andre Roberson	Russell Westbrook	Enes Kanter	5
		Steven Adams	Andre Roberson	Russell Westbrook	85
Kyle Singler	Enes Kanter	Kevin Durant	Randy Foye	Nick Collison	15
			Steven Adams	Randy Foye	7
	Randy Foye	Nick Collison	Russell Westbrook	Serge Ibaka	1
		Russell Westbrook	Serge Ibaka	Steven Adams	15
	Russell Westbrook	Serge Ibaka	Andre Roberson	Enes Kanter	3
			Steven Adams	Andre Roberson	7
Russell Westbrook	Andre Roberson	Serge Ibaka	Kevin Durant	Steven Adams	61
	Serge Ibaka	Andre Roberson	Enes Kanter	Kevin Durant	71
		Kevin Durant	Andre Roberson	Anthony Morrow	1
				Steven Adams	3
			Anthony Morrow	Randy Foye	4
			Randy Foye	Andre Roberson	4
			Steven Adams	Randy Foye	6
Serge Ibaka	Andre Roberson	Kyle Singler	Enes Kanter	Kevin Durant	1
		Russell Westbrook	Enes Kanter	Kyle Singler	14
			Kyle Singler	Enes Kanter	38
	Enes Kanter	Kyle Singler	Randy Foye	Kevin Durant	1
	Russell Westbrook	Enes Kanter	Kyle Singler	Randy Foye	27
	Steven Adams	Andre Roberson	Russell Westbrook	Kyle Singler	1

```
Name: h4, dtype: int64
```

Here, we see that the pairings of **Westbrook, Ibaka, Durant, Roberson, Adams** appears 3 times while **Durant, Ibaka, Adams, Roberson, West-**

brook appears 85 times. Similarly, the pairings of **Kanter, Singler, Foye, Durant, Collison** appears 18 times; as well as **Singler, Kanter, Durant, Foye, Collison** appearing 15 more times. We see of the 30 "different" line-ups for the Oklahoma City Thunder, there are really much less.

To perform this sort across columns, we extract the columns and dump the values to a collection of lists. We then sort each list and build a new dataframe that can be inserted back into the original dataframe.

```
>>> a = game.loc[0:,'a1':'a5'].values
>>> a.sort(axis=1)
>>> awaysort = pd.DataFrame(a,game.index,list(game.columns.values)[3:8])
>>> game.loc[0:,'a1':'a5'] = awaysort.loc[0:,'a1':'a5']
>>> onCourt = game.groupby(['a1','a2','a3','a4','a5'])
>>> onCourt['h4'].count()
a1              a2              a3                a4                a5
Andre Roberson  Anthony Morrow  Kevin Durant      Russell Westbrook  Serge Ibaka           1
                Enes Kanter     Kevin Durant      Kyle Singler       Serge Ibaka           1
                                                                     Steven Adams          5
                                                  Randy Foye         Russell Westbrook     1
                                                  Russell Westbrook  Serge Ibaka          82
                                Kyle Singler      Russell Westbrook  Serge Ibaka          55
                                Randy Foye        Russell Westbrook  Serge Ibaka           9
                Kevin Durant    Randy Foye        Russell Westbrook  Serge Ibaka           4
                                Russell Westbrook Serge Ibaka        Steven Adams        149
                Kyle Singler    Russell Westbrook Serge Ibaka        Steven Adams          8
Anthony Morrow  Kevin Durant    Randy Foye        Russell Westbrook  Serge Ibaka           4
Enes Kanter     Kevin Durant    Kyle Singler      Nick Collison      Randy Foye           78
                                                  Randy Foye         Serge Ibaka           1
                                                                     Steven Adams         16
                                Nick Collison     Randy Foye         Russell Westbrook    12
                Kyle Singler    Nick Collison     Randy Foye         Russell Westbrook     1
                                Randy Foye        Russell Westbrook  Serge Ibaka          27
Kevin Durant    Randy Foye      Russell Westbrook Serge Ibaka        Steven Adams          6
Kyle Singler    Nick Collison   Randy Foye        Russell Westbrook  Serge Ibaka           1
                Randy Foye      Russell Westbrook Serge Ibaka        Steven Adams         15
Name: h4, dtype: int64
```

Thus we have managed to sort and count all the actions in the NBA datafile; identifying the correct 20 rotation combinations. We perform the same tasks for the home team and then we can group by the different 10 player rotations between both teams. We find that there are 53 total rotation combinations on the court at any given point of the game.

```
>>> onCourt = games.groupby(['a1','a2','a3','a4','a5','h1','h2','h3','h4','h5'])
>>> onCourt['h4'].count()
>>> len(onCourt['h4'].count())
53
```

2.4 Counting Statistics

We can use the groupby function and knowledge of the dataframe to start counting statistics. For instance, let's count the number of assists in the March 12th Spurs - Thunder game. Since there is a column titled **assist** that labels the player who obtained an assist on a **shot** action, we can group by the assist column and perform a count operation.

```
>>> assists = game.groupby('assist')
>>> assists['assist'].count()
assist
Andre Roberson       1
Boris Diaw           2
Danny Green          1
Kawhi Leonard        3
Kevin Durant         8
LaMarcus Aldridge    3
Nick Collison        1
Patty Mills          3
Randy Foye           1
Russell Westbrook    7
Tim Duncan           1
Tony Parker          4
Name: assist, dtype: int64
```

Here, the groupby action only grabs the players who picked up assists. There are only 12 players in this case. Each of the 12 groups contain only the actions for which the assists took place. For instance, there is a column titled **shot_distance** which measures the distance from the basket at which the field goal was made. We can then calculate the average shot distance for which a player obtains an assist.

```
>>> assists['shot_distance'].mean()
assist
Andre Roberson       16.000000
Boris Diaw           19.000000
Danny Green           3.000000
Kawhi Leonard        15.333333
Kevin Durant          7.125000
LaMarcus Aldridge    12.666667
Nick Collison         0.000000
Patty Mills          14.666667
Randy Foye           18.000000
Russell Westbrook    11.857143
Tim Duncan           13.000000
Tony Parker          14.750000
Name: shot_distance, dtype: float64
```

As we can see, Collison passes to dunks only. Randy Foye and Boris Diaw kick out for long-range jumpers. So now let's look at building field goal percentages.

```
>>> events = game.groupby(['event_type'])
```

```
>>> shots = game.iloc[events.groups['shot']].groupby('player')
>>> shots['player'].count()
player
Andre Roberson        2
Boris Diaw            2
Danny Green           1
David West            4
Enes Kanter           4
Kawhi Leonard        10
Kevin Durant         11
Kevin Martin          1
Kyle Anderson         1
LaMarcus Aldridge     9
Patty Mills           1
Randy Foye            1
Russell Westbrook     5
Serge Ibaka           3
Steven Adams          5
Tim Duncan            5
Name: player, dtype: int64

>>> misses = game.iloc[events.groups['miss']].groupby('player')
>>> misses['player'].count()
player
Anthony Morrow        1
Boris Diaw            1
Danny Green           9
David West            6
Enes Kanter           2
Kawhi Leonard        14
Kevin Durant         14
Kyle Anderson         2
Kyle Singler          5
LaMarcus Aldridge     5
Manu Ginobili         3
Nick Collison         1
Patty Mills           5
Randy Foye            3
Russell Westbrook    11
Serge Ibaka          10
Steven Adams          3
Tim Duncan            2
Tony Parker           4
Name: player, dtype: int64
```

So we have a list of made field goals and missed field goals. We can then

form the field goal percentage by the usual makes divided by makes plus misses formula. However, we will obtain NaN values since there are players who do not exist on the made shot list.

```
>>> S = shots['player'].count()
>>> M = misses['player'].count()
>>> FGP = S/(S+M)
>>> FGP
player
Andre Roberson            NaN
Anthony Morrow            NaN
Boris Diaw           0.666667
Danny Green          0.100000
David West           0.400000
Enes Kanter          0.666667
Kawhi Leonard        0.416667
Kevin Durant         0.440000
Kevin Martin              NaN
Kyle Anderson        0.333333
Kyle Singler              NaN
LaMarcus Aldridge    0.642857
Manu Ginobili             NaN
Nick Collison             NaN
Patty Mills          0.166667
Randy Foye           0.250000
Russell Westbrook    0.312500
Serge Ibaka          0.230769
Steven Adams         0.625000
Tim Duncan           0.714286
Tony Parker               NaN
Name: player, dtype: float64
```

To correct for this, we use the **fillna** function in pandas to replace the NaN values. However, we first must perform a merge to get the NaN values to appear in the shots dataframe. If you have not noticed, the FGP dataframe above wiped Andre Roberson's 100 percent field goal percentage. Once the NaN values appear, we set each to zero and then perform two column creations: Total shots and FGP. Once we obtain these two new columns, we eliminate the misses column and obtain the resulting box-score like table.

```
>>> S.name = 'shot'
>>> M.name = 'miss'
>>> B = pd.concat([S,M],axis=1)
>>> B
                   shot  miss
Andre Roberson        2   NaN
Anthony Morrow      NaN     1
```

```
Boris Diaw            2     1
Danny Green           1     9
David West            4     6
Enes Kanter           4     2
Kawhi Leonard        10    14
Kevin Durant         11    14
Kevin Martin          1   NaN
Kyle Anderson         1     2
Kyle Singler        NaN     5
LaMarcus Aldridge     9     5
Manu Ginobili       NaN     3
Nick Collison       NaN     1
Patty Mills           1     5
Randy Foye            1     3
Russell Westbrook     5    11
Serge Ibaka           3    10
Steven Adams          5     3
Tim Duncan            5     2
Tony Parker         NaN     4

>>> B['shot'] = B['shot'].fillna(0)
>>> B['miss'] = B['miss'].fillna(0)
>>> B['total'] = B['shot']+B['miss']
>>> B['FGP'] = B['shot'] / B['total']
>>> B = B.drop('miss',1)
>>> B
                   shot  total       FGP
Andre Roberson        2      2  1.000000
Anthony Morrow        0      1  0.000000
Boris Diaw            2      3  0.666667
Danny Green           1     10  0.100000
David West            4     10  0.400000
Enes Kanter           4      6  0.666667
Kawhi Leonard        10     24  0.416667
Kevin Durant         11     25  0.440000
Kevin Martin          1      1  1.000000
Kyle Anderson         1      3  0.333333
Kyle Singler          0      5  0.000000
LaMarcus Aldridge     9     14  0.642857
Manu Ginobili         0      3  0.000000
Nick Collison         0      1  0.000000
Patty Mills           1      6  0.166667
Randy Foye            1      4  0.250000
Russell Westbrook     5     16  0.312500
Serge Ibaka           3     13  0.230769
Steven Adams          5      8  0.625000
```

```
Tim Duncan              5       7  0.714286
Tony Parker             0       4  0.000000
```

We finish this section on computing basic statistics by calculating the amount of time played for each player.

```
>>> a = game.loc[0:,'a1':'a5'].values
>>> a.sort(axis=1)
>>> awaysort = pd.DataFrame(a,game.index,list(game.columns.values)[3:8])
>>> game.loc[0:,'a1':'a5'] = awaysort.loc[0:,'a1':'a5']
>>> onCourt = game.groupby(['a1','a2','a3','a4','a5'])
>>> len(onCourt.groups.keys())
>>> for i in range(len(onCourt.groups.keys())):
...     print onCourt.groups.keys()[i], onCourt.get_group(onCourt.groups.keys()[i])
['play_length'].str[6:].convert_objects(convert_numeric=True).sum()
...

('Enes Kanter', 'Kevin Durant', 'Kyle Singler', 'Nick Collison', 'Randy Foye') 454.0
('Anthony Morrow', 'Kevin Durant', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka') 15
('Kyle Singler', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka', 'Steven Adams') 73
('Kyle Singler', 'Nick Collison', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka') 0
('Andre Roberson', 'Enes Kanter', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka') 101
('Andre Roberson', 'Enes Kanter', 'Kevin Durant', 'Russell Westbrook', 'Serge Ibaka') 503
('Andre Roberson', 'Enes Kanter', 'Kevin Durant', 'Kyle Singler', 'Serge Ibaka') 0
('Andre Roberson', 'Enes Kanter', 'Kevin Durant', 'Kyle Singler', 'Steven Adams') 40
('Enes Kanter', 'Kyle Singler', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka') 114
('Andre Roberson', 'Enes Kanter', 'Kyle Singler', 'Russell Westbrook', 'Serge Ibaka') 312
('Andre Roberson', 'Enes Kanter', 'Kevin Durant', 'Randy Foye', 'Russell Westbrook') 0
('Enes Kanter', 'Kyle Singler', 'Nick Collison', 'Randy Foye', 'Russell Westbrook') 0
('Andre Roberson', 'Kevin Durant', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka') 5
('Enes Kanter', 'Kevin Durant', 'Kyle Singler', 'Randy Foye', 'Serge Ibaka') 0
('Kevin Durant', 'Randy Foye', 'Russell Westbrook', 'Serge Ibaka', 'Steven Adams') 5
('Enes Kanter', 'Kevin Durant', 'Kyle Singler', 'Randy Foye', 'Steven Adams') 78
('Andre Roberson', 'Kevin Durant', 'Russell Westbrook', 'Serge Ibaka', 'Steven Adams') 1057.0
('Andre Roberson', 'Kyle Singler', 'Russell Westbrook', 'Serge Ibaka', 'Steven Adams') 44
('Enes Kanter', 'Kevin Durant', 'Nick Collison', 'Randy Foye', 'Russell Westbrook') 79
('Andre Roberson', 'Anthony Morrow', 'Kevin Durant', 'Russell Westbrook', 'Serge Ibaka') 0


>>> tots = 0
>>> for i in range(len(onCourt.groups.keys())):
...     tots = tots + onCourt.get_group(onCourt.groups.keys()[i])
['play_length'].str[6:].convert_objects(convert_numeric=True).sum()
...
>>> tots
2880.0
>>> 2880/60
48

>>> playerTimes = {'player':'time'}
>>> for i in range(len(onCourt.groups.keys())):
...     for j in range(5):
...             if onCourt.groups.keys()[i][j] in playerTimes:
...                     playerTimes[onCourt.groups.keys()[i][j]] += onCourt.get_group(onCourt.groups.keys()[i])
['play_length'].str[6:].convert_objects(convert_numeric=True).sum()
...             else:
...                     playerTimes[onCourt.groups.keys()[i][j]] = onCourt.get_group(onCourt.groups.keys()[i])
['play_length'].str[6:].convert_objects(convert_numeric=True).sum()
...
>>> playerTimes
{'Anthony Morrow': 15, 'Enes Kanter': 1681.0, 'Kevin Durant': 2236.0, 'Serge Ibaka': 2229.0,
'Russell Westbrook': 2308.0, 'Andre Roberson': 2062.0, 'player': 'time', 'Randy Foye': 924.0,
'Nick Collison': 533.0, 'Steven Adams': 1297.0, 'Kyle Singler': 1115.0}

>>> del playerTimes['player']
>>> playerTimes
{'Anthony Morrow': 15, 'Enes Kanter': 1681.0, 'Kevin Durant': 2236.0, 'Serge Ibaka': 2229.0,
```

```
'Russell Westbrook': 2308.0, 'Andre Roberson': 2062.0, 'Randy Foye': 924.0, 'Nick Collison': 533.0,
'Steven Adams': 1297.0, 'Kyle Singler': 1115.0}

>>> for key, value in pt.items():
...     pt[key] = str(math.floor(value/60))[:-2] + ':' + str(value\%60)
...
>>> pt
{'Anthony Morrow': '0:15', 'Enes Kanter': '28:1.0', 'Kevin Durant': '37:16.0', 'Serge Ibaka': '37:9.0',
'Russell Westbrook': '38:28.0', 'Andre Roberson': '34:22.0', 'Randy Foye': '15:24.0',
'Nick Collison': '8:53.0', 'Steven Adams': '21:37.0', 'Kyle Singler': '18:35.0'}

>>> for player in pt:
...     print player, pt[player]
...

Anthony Morrow 0:15
Enes Kanter 28:1.0
Kevin Durant 37:16.0
Serge Ibaka 37:9.0
Russell Westbrook 38:28.0
Andre Roberson 34:22.0
Randy Foye 15:24.0
Nick Collison 8:53.0
Steven Adams 21:37.0
Kyle Singler 18:35.0
```

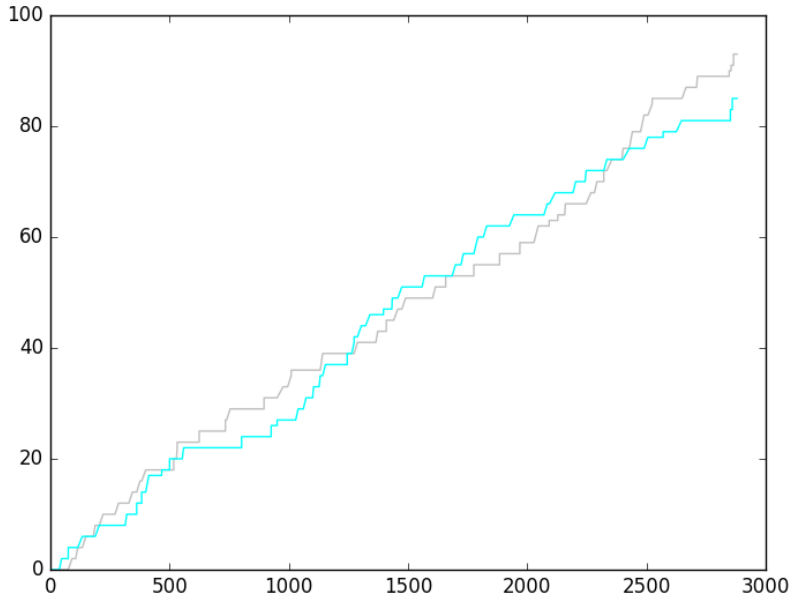
Comparing to the official stats (note that the data file comes from the NBA) we see that:

Player	Python	NBA Page
Russell Westbrook	38:28	38:14
Kevin Durant	37:16	37:13
Serge Ibaka	37:09	37:09
Andre Roberson	34:22	34:22
Enes Kanter	28:01	28:01
Steven Adams	21:37	21:36
Kyle Singler	18:35	18:49
Randy Foye	15:24	15:28
Nick Collison	8:53	8:53
Anthony Morrow	0:15	0:15
Total	48:00	48:00

This suggests that there are corrections between the NBA data file and the official statistics. This may be something as subtle as counting seconds based on time before/after substitutions.

3 Visualization

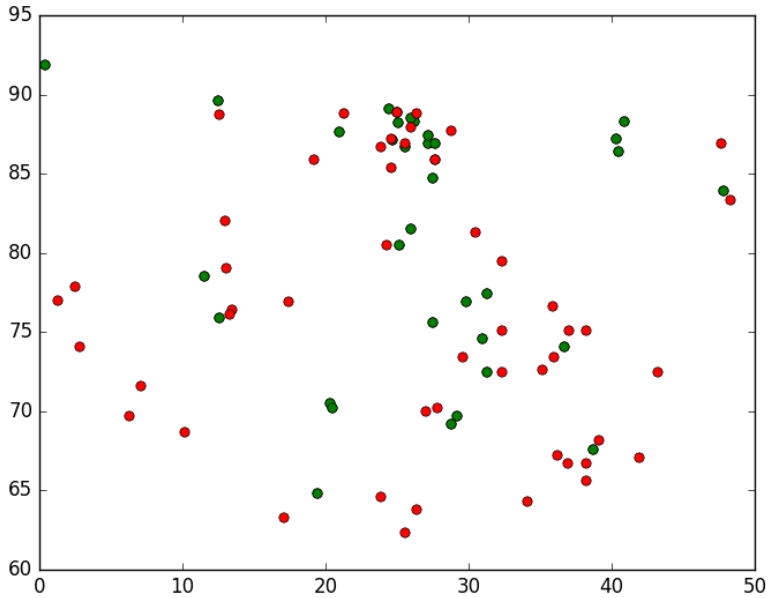
In this section, we take a quick look at displaying some plots of data obtained in Pandas data frames. In order to do this, we call the **matplotlib.pyplot** package and utilize the plotting tools within. A simple example is to build a score flow of the Spurs - Thunder game. In this case, we take the **remaining_time** variable and convert its values to seconds. We then match the Pandas series to the



away_score and **home_score** variables. From there, we simply call the **plot** and **show** functions contained in the **matplotlib.pyplot** package. Note that we must plot all sections of the graph before showing the image!

```
>>> time = (11-game['remaining_time'].str[3:5].convert_objects(convert_numeric=True))*60
 + (60-game['remaining_time'].str[6:8].convert_objects(convert_numeric=True))
  + (game['period'].convert_objects(convert_numeric=True) - 1)*720
>>> away = game['away_score'].convert_objects(convert_numeric=True)
>>> home = game['home_score'].convert_objects(convert_numeric=True)
>>> plt.plot(time,home,'silver')
[<matplotlib.lines.Line2D object at 0x116bb8390>]
>>> plt.plot(time,away,'>>> plt.plot(time,away,'cyan')
[<matplotlib.lines.Line2D object at 0x116bb8910>]
>>> plt.show()
```

A second plot we build, in a simple way, is the shot chart for the San Antonio Spurs. Here, we can form a **groupby** on the **team** and **event_type** columns in the data set. We will partition the set into Spurs actions, labeled **SAS**, and into Thunder actions, labeled **OKC**. Referencing both **shot** and **miss** and color coding as green and red, respectively, we are able to display a basic shot chart. From the dispersion of shots, it is obvious where the basket is located and the discriminating three-point line, separating three-point field goals and two-point field goals. Note that in the March 12th game, the Spurs (despite



winning 93-85) shot a paltry 4-for-24 from behind the line.

```
>>> team = game.groupby(['team','event_type'])
>>> team.get_group(('SAS', 'shot'))
>>> team.get_group(('SAS', 'shot')).loc[0:,'converted_x':'converted_y']
>>> plt.plot(team.get_group(('SAS', 'shot')).loc[0:,'converted_x'],
team.get_group(('SAS', 'shot')).loc[0:,'converted_y'],'go')
[<matplotlib.lines.Line2D object at 0x10ece0390>]
>>> plt.plot(team.get_group(('SAS', 'miss')).loc[0:,'converted_x'],
team.get_group(('SAS', 'miss')).loc[0:,'converted_y'],'ro')
[<matplotlib.lines.Line2D object at 0x10ece0d90>]
>>> plt.show()
```