

Phil Bingham's Data Science Portfolio (<https://hexhamallstar.github.io/>)

A collection of my personal data science/machine learning work.

Sat 17 March 2018

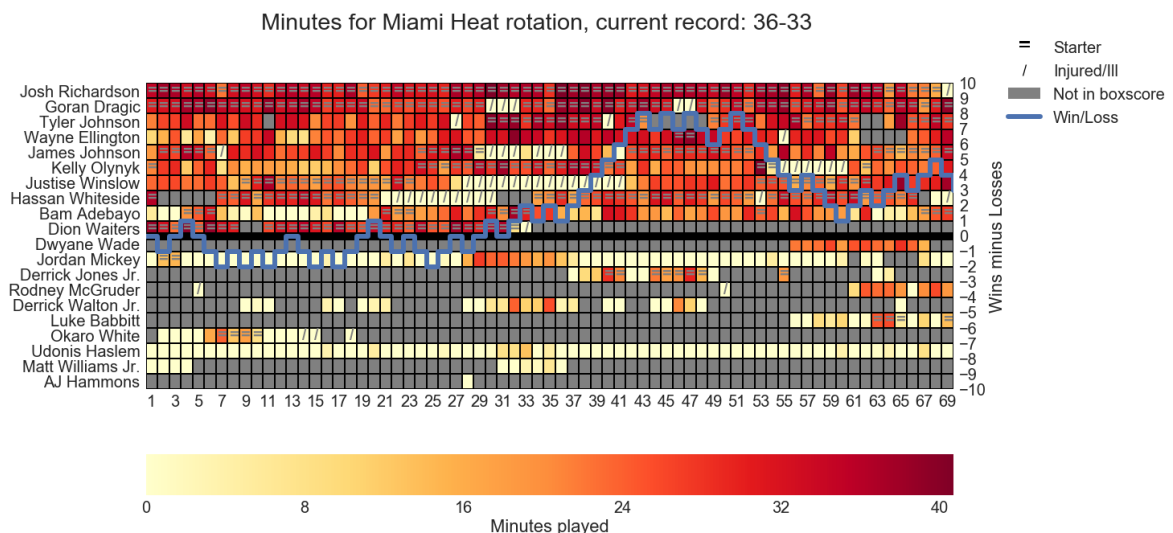
NBA Webscraping & Visualisation (<https://hexhamallstar.github.io/nbaviz.html>)

Posted by Phil Bingham (<https://hexhamallstar.github.io/author/phil-bingham.html>) in Notebooks

(<https://hexhamallstar.github.io/category/notebooks.html>)

There are a lot of large open-source datasets available online for building machine learning models which is excellent from a learning standpoint however it means that pretty much every one of these common datasets has been explored in great depth. I therefore decided that I wanted to create my own dataset to use for analysis/visualisation. In this notebook I will explain how I created the dataset and will create a method for plotting a complex visualisation for any team in the dataset.

The visualisation that I want to create will display information on all of a teams players' minutes for each game (if they played), their reason for not playing (if they didn't play), an indication of whether a particular player was in the starting lineup and also the teams win/loss record as the season progresses. Here is an example of the final chart I'm aiming to plot.



([images/miami-heat.png](#))

This notebook contains the following content:

- [1. Selenium Webscraping](#)
- [2. Importing and Cleaning Data from SQL](#)
- [3. Flexible Data Visualisation](#)

1. Selenium Webscraping

Selenium is a Python library that allows the user to automate the actions of one or more web-browser instance(s). In traditional webscraping we can use the requests module to get the html code of a given web-page url, however for websites that use interactive elements (like <http://stats.nba.com/> (<http://stats.nba.com/>)) will have placeholder variable names in the html code rather than the actual values that we want.

Selenium allows us to actually load the web-page and, once the interactive elements have loaded, download a copy of the currently rendered html (rather than the template code). An example of the difference is shown below. This is an example of the boxscore for a team in a particular match:

Milwaukee Bucks																				
PLAYER	MIN	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	FT%	OREB	DREB	REB	AST	TOV	STL	BLK	PF	PTS	+/-
Khris Middleton ^F	34:23	8	16	50.0	4	6	66.7	2	2	100	0	3	3	5	3	0	0	4	22	-12
Giannis Antetokounmpo ^F	40:27	16	29	55.2	1	3	33.3	5	5	100	1	9	10	7	2	3	0	4	38	10
John Henson ^C	29:35	4	9	44.4	0	0	0.0	0	0	0.0	2	7	9	1	0	1	1	2	8	-12
Tony Snell ^G	17:42	1	1	100	0	0	0.0	0	0	0.0	0	0	0	1	0	1	0	1	2	3
Eric Bledsoe ^G	31:32	8	16	50.0	3	7	42.9	1	2	50.0	1	1	2	8	2	4	1	2	20	-3
Sterling Brown	9:32	0	2	0.0	0	1	0.0	0	0	0.0	0	0	0	0	0	0	0	2	0	-19
Brandon Jennings	8:31	1	3	33.3	0	1	0.0	0	0	0.0	0	1	1	1	1	0	0	2	2	-13
Jabari Parker	22:21	6	10	60.0	1	3	33.3	1	2	50.0	1	3	4	2	1	0	1	2	14	-12
Tyler Zeller	11:06	0	3	0.0	0	0	0.0	0	0	0.0	2	2	4	0	0	1	1	1	0	2
Jason Terry	24:08	2	4	50.0	1	3	33.3	0	0	0.0	0	1	1	2	1	2	1	0	5	4
Shabazz Muhammad	6:54	3	5	60.0	0	0	0.0	0	1	0.0	1	1	2	1	0	0	0	0	6	9
Thon Maker	3:49	0	0	0.0	0	0	0.0	0	0	0.0	1	0	1	0	1	0	0	0	0	-2
D.J. Wilson	DNP - COACH'S DECISION																			
Totals:		49	98	50.0	10	24	41.7	9	12	75.0	9	28	37	28	11	12	5	20	117	-9

([images/boxscore.png](#))

The page source for this table is a reference to an object, and we therefore can't get any information out of it:

```
<div class="columns small-12">
  <a ng-href="/game/{{ GameID }}/{{ currentSearchString }}" data-section="boxscore">Box Score</a>
```

([boxscore source html.png](#))

Whereas if we use Firefox' built in 'Inspect Element' button (on right-click) shows the following:

	MIN	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	FT%	OREB	DREB	REB	AST	TOV	ST
Giannis Antetokounmpo F	40:27	16	29	55.2	1	3	33.3	5	5	100	1	9	10	7	2	
John Henson G	29:35	4	9	44.4	0	0	0.0	0	0	0.0	2	7	9	1	0	
Tony Snell G	17:42	1	1	100	0	0	0.0	0	0	0.0	0	0	0	1	0	
Eric Bledsoe G	31:32	8	16	50.0	3	7	42.9	1	2	50.0	1	1	2	8	2	
Sterling Brown	9:32	0	2	0.0	0	1	0.0	0	0	0.0	0	0	0	0	0	
Brandon Jennings	8:31	1	3	33.3	0	1	0.0	0	0	0.0	0	1	1	1	1	
Jabari Parker	22:21	6	10	60.0	1	3	33.3	1	2	50.0	1	3	4	2	1	
Tyler Zeller	11:06	0	3	0.0	0	0	0.0	0	0	0.0	2	2	4	0	0	
Jason Terry	24:08	2	4	50.0	1	3	33.3	0	0	0.0	0	1	1	2	1	
Shabazz Muhammad	6:54	3	5	60.0	0	0	0.0	0	1	0.0	1	1	2	1	0	
Thon Maker	3:49	0	0	0.0	0	0	0.0	0	0	0.0	1	0	1	0	1	
D.J. Wilson																

(images/boxscore_inspect_element.png)

We can see that the actual information that we wanted is only viewable if we render the page, and therefore we need to use Selenium. I have uploaded my whole webscraper [here](https://github.com/HexhamAllstar/NBA-Scraper) (<https://github.com/HexhamAllstar/NBA-Scraper>), feel free to download it and give it a try.

The webscraper works on the following basis:

1. For a specific day, the NBA scores page has a build-able URL (i.e. for 14th March 2018 the URL would be <http://stats.nba.com/scores/03/14/2018> (<http://stats.nba.com/scores/03/14/2018>))
2. On the scores page for a given day, there is an overview for each game and a button that can take us to the boxscore for that game.
3. Therefore for each date, we can use Selenium to get the team names and scores and a URL for the boxscore of each game.
4. We then send a Selenium instance to each of the boxscore URL's to grab the data out of the table.
5. Insert the scraped data into a local SQL database using the python sqlite3 library, creating 2 tables called results and boxscores. Results contains just the game summary information and boxscores contains the actual player specific stats.

Now that we have a local database, we can query it as much as we like and get data in a consistent, usable format.

2. Importing and Cleaning Data from SQL

Results data manipulation/cleaning

In [1]:

```
import pandas as pd
import numpy as np
import backend
# backend is the name of a .py file included with the webscraper that provides bas
```

```
In [2]: results_data = backend.retrieve_all_results() # command 'SELECT * from results'
        results_data.head()
```

Out[2]:

	GameID	GameDate	HomeTeam	HomeScore	AwayTeam	AwayScore
0	0021701005	2018-03-13 00:00:00	Indiana Pacers	101	Philadelphia 76ers	98
1	0021701006	2018-03-13 00:00:00	Minnesota Timberwolves	116	Washington Wizards	111
2	0021701007	2018-03-13 00:00:00	Oklahoma City Thunder	119	Atlanta Hawks	107
3	0021701008	2018-03-13 00:00:00	Toronto Raptors	116	Brooklyn Nets	102
4	0021701009	2018-03-13 00:00:00	Dallas Mavericks	110	New York Knicks	97

```
In [3]: results_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028 entries, 0 to 1027
Data columns (total 6 columns):
GameID      1028 non-null object
GameDate    1028 non-null object
HomeTeam     1028 non-null object
HomeScore    1028 non-null object
AwayTeam     1028 non-null object
AwayScore    1028 non-null object
dtypes: object(6)
memory usage: 48.3+ KB
```

We can see that the results data has a unique ID number, a date, name and scores of the home and away teams. We have information for 1028 games, and all rows have information in i.e. no null values. Let's take a look at the teams that are present.

```
In [4]: results_data['HomeTeam'].nunique()
```

Out[4]: 32

Interestingly we get 32 unique home teams in the data. That seems strange because there are only 30 teams in the NBA. Let's see what teams are there.

```
In [5]: results_data['HomeTeam'].unique()

Out[5]: array(['Indiana Pacers', 'Minnesota Timberwolves',
              'Oklahoma City Thunder', 'Toronto Raptors', 'Dallas Mavericks',
              'LA Clippers', 'Charlotte Hornets', 'Orlando Magic',
              'Detroit Pistons', 'Cleveland Cavaliers', 'Denver Nuggets',
              'San Antonio Spurs', 'Milwaukee Bucks', 'Sacramento Kings',
              'Miami Heat', 'Chicago Bulls', 'Golden State Warriors',
              'Utah Jazz', 'Houston Rockets', 'Philadelphia 76ers',
              'Phoenix Suns', 'Washington Wizards', 'Memphis Grizzlies',
              'Atlanta Hawks', 'New York Knicks', 'Los Angeles Lakers',
              'Brooklyn Nets', 'Boston Celtics', 'New Orleans Pelicans',
              'Portland Trail Blazers', 'Team LeBron', ''], dtype=object)
```

Notice that we get 'Team LeBron' and '' at the end. Team LeBron is the name of one of the teams from this years All-star game, and is therefore not a real team. On further inspection it looks like the blank team name comes from the USA vs World game also during the all-star weekend. Let's get a list of valid team names:

```
In [6]: team_names = list(results_data['HomeTeam'].unique())
        team_names.remove('Team LeBron')
        team_names.remove(' ')
        print(team_names)

['Indiana Pacers', 'Minnesota Timberwolves', 'Oklahoma City Thunder', 'Toronto Ra
```

Now we want to look at a specific team. The first step is to isolate games that have our desired team ('Miami Heat' in this example):

```
In [7]: def get_team_games(team_name):
        """Takes in a string of a team name and returns a dataframe of all games this
        if team_name in team_names: #if the entered team name is valid
            df = results_data[(results_data['HomeTeam'] == team_name) | (results_data[
            df.sort_values(by='GameDate', ascending=True, inplace=True)
            df.reset_index(drop=True, inplace=True)
            return df
        else:
            print('That name is not valid, please enter a valid team name (case sensit

        miami_games = get_team_games('Miami Heat')
        miami_games.head()
```

```
Out[7]:
```

	GameID	GameDate	HomeTeam	HomeScore	AwayTeam	AwayScore
0	0021700005	2017-10-18 00:00:00	Miami Heat	109	Orlando Magic	116
1	0021700029	2017-10-21 00:00:00	Indiana Pacers	108	Miami Heat	112
2	0021700042	2017-10-23 00:00:00	Atlanta Hawks	93	Miami Heat	104
3	0021700059	2017-10-25 00:00:00	San Antonio Spurs	117	Miami Heat	100
4	0021700078	2017-10-28 00:00:00	Boston Celtics	96	Miami Heat	90

Now we have a dataframe of games just for the Miami Heat. If we want to show how the teams record changes throughout the season we need to add a new column which indicates whether Miami won or not:

```
In [8]: def calc_wins(row, team_name):
        """This function takes in a row of a dataframe and a team name and returns a c
        equal to 1 if the specified team won and -1 if the specified team lost."""
        # first check to see if the team name is valid and in this dataframe
        if (team_name in team_names) & (team_name in [row['HomeTeam'], row['AwayTeam']]):
            # then calculate whether the team of interest won or not
            if (row['HomeTeam'] == team_name) & (int(row['HomeScore']) > int(row['Away
            row['Win'] = 1
            elif (row['AwayTeam'] == team_name) & (int(row['AwayScore']) > int(row['Ho
            row['Win'] = 1
            else:
                row['Win'] = -1
            return row
        else:
            print('Invalid team name')

        def add_win_col(df, team_name):
            df = df.apply(calc_wins, args=(team_name,), axis=1)
            return df

        miami_games = add_win_col(miami_games, 'Miami Heat')
        miami_games.head(10)
```

Out[8]:

	GameID	GameDate	HomeTeam	HomeScore	AwayTeam	AwayScore	Win
0	0021700005	2017-10-18 00:00:00	Miami Heat	109	Orlando Magic	116	-1
1	0021700029	2017-10-21 00:00:00	Indiana Pacers	108	Miami Heat	112	1
2	0021700042	2017-10-23 00:00:00	Atlanta Hawks	93	Miami Heat	104	1
3	0021700059	2017-10-25 00:00:00	San Antonio Spurs	117	Miami Heat	100	-1
4	0021700078	2017-10-28 00:00:00	Boston Celtics	96	Miami Heat	90	-1
5	0021700093	2017-10-30 00:00:00	Minnesota Timberwolves	125	Miami Heat	122	-1
6	0021700110	2017-11-01 00:00:00	Chicago Bulls	91	Miami Heat	97	1
7	0021700127	2017-11-03 00:00:00	Miami Heat	94	Denver Nuggets	95	-1
8	0021700137	2017-11-05 00:00:00	Miami Heat	104	LA Clippers	101	1
9	0021700148	2017-11-06 00:00:00	Miami Heat	80	Golden State Warriors	97	-1

We can confirm that, at least for the first 10 rows, the Win column correctly shows a -1 when Miami lost and +1 when Miami won. The reason we have used +/- 1 is so that we can do a cumulative sum and find

out how many more games the team has won than lost. We will call this the '+/-' of the team:

```
In [9]: def add_plus_minus(df):
        """Takes in a df with 'Win' column (+1 for win, -1 for loss) and adds a
        '+/-' column showing the cumulative sum of the wins column"""
        df['+/-'] = df['Win'].cumsum()
        return df

        miami_games = add_plus_minus(miami_games)
        miami_games.head()
```

Out[9]:

	GameID	GameDate	HomeTeam	HomeScore	AwayTeam	AwayScore	Win	+/-
0	0021700005	2017-10-18 00:00:00	Miami Heat	109	Orlando Magic	116	-1	-1
1	0021700029	2017-10-21 00:00:00	Indiana Pacers	108	Miami Heat	112	1	0
2	0021700042	2017-10-23 00:00:00	Atlanta Hawks	93	Miami Heat	104	1	1
3	0021700059	2017-10-25 00:00:00	San Antonio Spurs	117	Miami Heat	100	-1	0
4	0021700078	2017-10-28 00:00:00	Boston Celtics	96	Miami Heat	90	-1	-1

Now that we have the '+/-' column, we would like to plot it as the season progresses. We could plot it against the date, but with the uneven spacing between games it wouldn't look quite right. For this reason we add a 'GameNumber' column which just gives each game a sequential number:

```
In [10]: def add_game_number(df):
        """Adds a column 'GameNumber' to a df with games sorted by date (oldest game f
        df['GameNumber'] = df.index + 1
        return df

        miami_games = add_game_number(miami_games)
        miami_games.head()
```

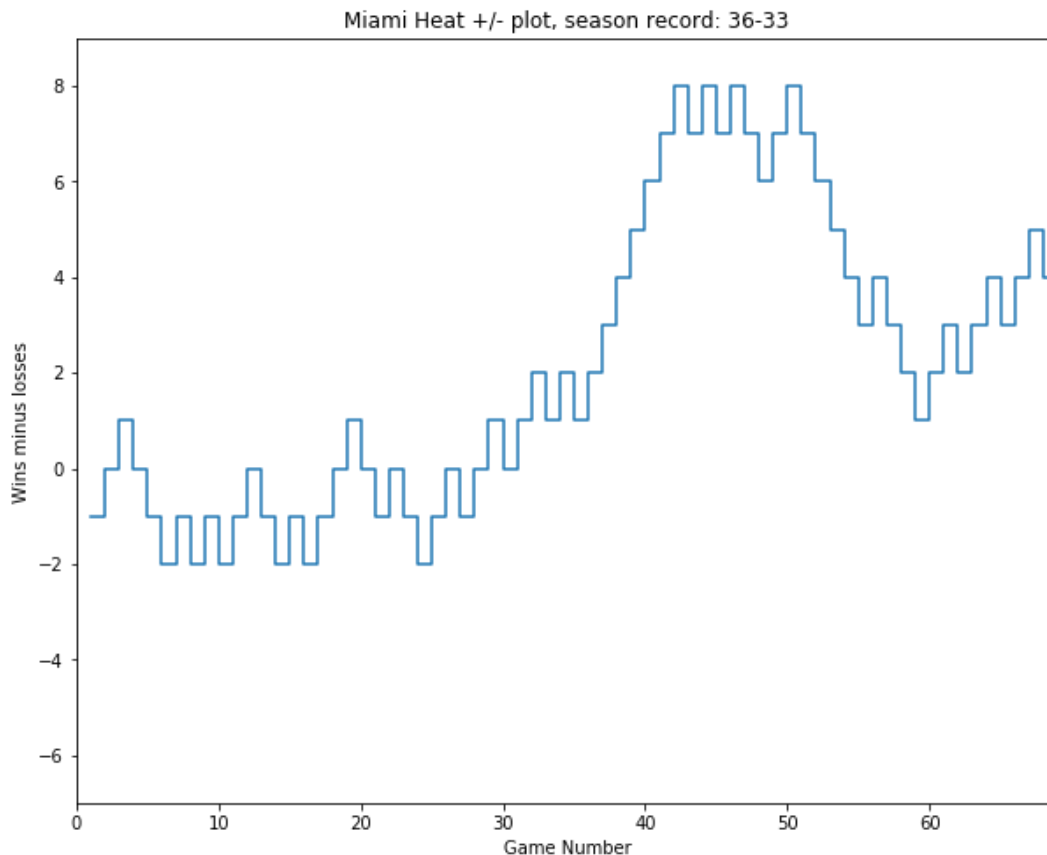
Out[10]:

	GameID	GameDate	HomeTeam	HomeScore	AwayTeam	AwayScore	Win	+/-	GameNumber
0	0021700005	2017-10-18 00:00:00	Miami Heat	109	Orlando Magic	116	-1	-1	1
1	0021700029	2017-10-21 00:00:00	Indiana Pacers	108	Miami Heat	112	1	0	2
2	0021700042	2017-10-23 00:00:00	Atlanta Hawks	93	Miami Heat	104	1	1	3
3	0021700059	2017-10-25 00:00:00	San Antonio Spurs	117	Miami Heat	100	-1	0	4
4	0021700078	2017-10-28 00:00:00	Boston Celtics	96	Miami Heat	90	-1	-1	5


```
In [11]: # import pyplot and set it to plot inline
import matplotlib.pyplot as plt
%matplotlib inline

def plus_minus_plot(df, team_name):
    """Takes in a dataframe with a '+/-' column and a 'GameNumber' column and plot
    be input as a string to use in the title of the plot."""
    fig, ax= plt.subplots(figsize=(10,8)) # create a figure of the correct width a
    ax.plot(df['GameNumber'], df['+/-'],drawstyle='steps-post') #draw our line on
    # the drawstyle keyword sets the 'square-wave' style
    # set limits based on the maximum absolute value in the column
    ax.set_ylim(-(df['+/-'].abs().max()+1),(df['+/-'].abs().max()+1))
    ax.set_xlim((0,df['GameNumber'].max()))
    ax.set_xlabel('Game Number')
    ax.set_ylabel('Wins minus losses')
    # calculate how many losses the team has so far
    losses = (len(list(df['GameNumber'])) - list(df['+/-'])[-1])/2
    # calculate how many wins the team has so far
    wins = losses + list(df['+/-'])[-1]
    # create a title that has the team name and their wins-losses in it
    ax.set_title(team_name + ' +/- plot, season record: ' + str(int(wins)) + '-' +

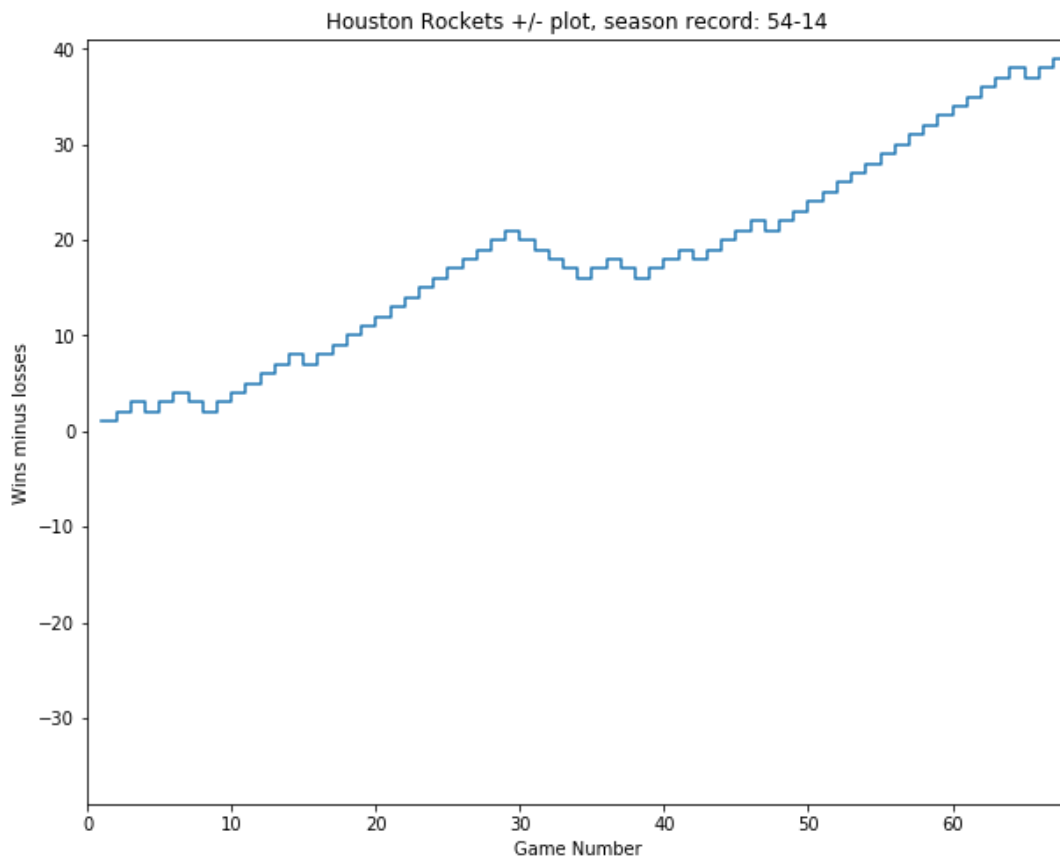
plus_minus_plot(miami_games, 'Miami Heat')
```



This is all of the information that I wish to display from the results table. All of the previous functions can be combined into a pipeline, allowing us to generate the above graph for any team just by running one line of code.

```
In [12]: def results_pipeline(team_name):
  """Takes in a string of the team name (case sensitive) and plots the +/- of th
  """
  # First check if the team name is valid
  if team_name in team_names:
      # then call the sequence of functions defined above
      df = get_team_games(team_name)
      df = df.apply(calc_wins, args=(team_name,), axis=1)
      df = add_plus_minus(df)
      df = add_game_number(df)
      plus_minus_plot(df, team_name)
  return df

houston_games = results_pipeline('Houston Rockets')
```



Now we can generate the processed dataframe and +/- plot with just a single line for any team. This means that on a given day I can simply update the database using my webscraper and then run the above line using a for loop, saving the resulting plot into a directory for every team in the list of valid teams. The rest of the data we need comes from the boxscores table.

Boxscore data

In [13]:

boxscore_data = backend.retrieve_all_boxscores() # SQL statement 'SELECT * from bo
boxscore_data.head()

Out[13]:

	Player Name	Min	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	...	AST	TOV	STL	BLK	PF	PTS	+/-	Team	Starter
0	Bojan Bogdanovic	25:58	11	11	9.1	0	5	0.0	0	1	...	3	1	0	0	2	2	-13	Indiana Pacers	1
1	Thaddeus Young	31:16	7	11	63.6	0	1	0.0	5	5	...	2	1	2	0	1	19	2	Indiana Pacers	1
2	Myles Turner	30:12	9	12	75.0	2	4	50.0	5	6	...	0	2	1	0	5	25	13	Indiana Pacers	1
3	Victor Oladipo	33:27	4	21	19.0	1	4	25.0	2	3	...	3	1	2	1	4	11	14	Indiana Pacers	1
4	Cory Joseph	33:10	5	8	62.5	1	2	50.0	2	2	...	5	1	3	0	1	13	-1	Indiana Pacers	1

5 rows × 24 columns

We see that each row of the dataframe corresponds to the performance statistics of a single player for a team in a particular game specified by GameID. Let's check for completeness of the data.

```
In [14]: boxscore_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25885 entries, 0 to 25884
Data columns (total 24 columns):
Player Name    25885 non-null object
Min            25885 non-null object
FGM            25885 non-null object
FGA            21877 non-null object
FG%            21877 non-null object
3PM            21877 non-null object
3PA            21877 non-null object
3P%            21877 non-null object
FTM            21877 non-null object
FTA            21877 non-null object
FT%            21877 non-null object
OREB           21877 non-null object
DREB           21877 non-null object
REB            21877 non-null object
AST            21877 non-null object
TOV            21877 non-null object
STL            21877 non-null object
BLK            21877 non-null object
PF             21877 non-null object
PTS            21877 non-null object
+/-            21877 non-null object
Team           25885 non-null object
Starter        25885 non-null int64
GameID         25885 non-null object
dtypes: int64(1), object(23)
memory usage: 4.7+ MB
```

We have 25885 individual entries, however only 21877 of these rows are complete, the others are missing data. It's also worth noting that currently all of the numerical stat columns are strings, not integers or floats. We will need to convert these later for plotting. Let's take a look at a missing row.

```
In [15]: boxscore_data[pd.isnull(boxscore_data['FGA'])].head()
```

```
Out[15]:
```

	Player Name	Min	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	...	AST	TOV	STL	BLK	PF	PTS	+/-
11	TJ Leaf	DNP - Coach's Decision	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None
12	Joe Young	DNP - Coach's Decision	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None
22	Justin Anderson	DNP - Coach's Decision	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None
23	Richaun Holmes	DNP - Coach's Decision	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None
24	Timothe Luwawu-Cabarrot	DNP - Coach's Decision	None	None	None	None	None	None	None	None	...	None	None	None	None	None	None	None

5 rows × 24 columns

```
In [16]: boxscore_data[pd.isnull(boxscore_data['FGA'])]['Min'].unique()
```

```
Out[16]: array([''], dtype=object)
```

It looks like, even though these players did not play they still appear in the boxscore for the team. The 'Min' column has an empty string in it and every stat column except 'FGM' is empty. The 'FGM' column contains the reasons for a player not playing.

```
In [17]: boxscore_data[pd.isnull(boxscore_data['FGA'])]['FGM'].unique()
```

```
Out[17]: array(['DNP - Coach's Decision',
        'DNP - Injury/Illness',
        'DND - Injury/Illness',
        'NWT - Injury/Illness',
        'DND - Rest',
        'NWT - Personal',
        'NWT - Suspended',
        'NWT - Trade Pending',
        'NWT - Rest',
        'Inactive - Injury/Illness'], dtype=object)
```

We can see that there are various possibilities for the reason a player didn't play. For example, if somebody played zero minutes in a game due to the coach just not putting them into the game, this

should be reflected differently in our plot than somebody not being played due to injury/illness. First we will clean the data for a specific team and create a pipeline, similar to what we did for the results data.

```
In [18]: def get_team_boxscores(team_name):
        """For a given team name string, returns a dataframe containing all boxscores
        from the team."""
        # check validity of team name
        if team_name in team_names:
            df = boxscore_data[boxscore_data['Team'] == team_name]
            return df
        else:
            print('Invalid team name')

        miami_boxscores = get_team_boxscores('Miami Heat')
        miami_boxscores.head()
```

Out[18]:

	Player Name	Min	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	...	AST	TOV	STL	BLK	PF	PPTS	+/-	Team	Starter
351	Josh Richardson	16:04	2	6	33.3	0	1	0.0	0	0	...	1	0	0	0	4	4	2	Miami Heat	1
352	James Johnson	24:02	3	5	60.0	0	1	0.0	2	4	...	4	2	0	0	2	8	-13	Miami Heat	1
353	Bam Adebayo	24:40	1	10	10.0	0	0	0.0	2	2	...	3	3	1	1	4	4	-8	Miami Heat	1
354	Tyler Johnson	34:48	6	15	40.0	4	7	57.1	1	2	...	2	0	0	0	2	17	-21	Miami Heat	1
355	Goran Dragic	30:45	10	17	58.8	0	3	0.0	3	3	...	2	2	1	0	4	23	-14	Miami Heat	1

5 rows × 24 columns

For the specific plots that we are trying to create, the only stats that we need are a players minutes and whether they started the game or not (indicated by the 'Starter' column). We also need to keep the FGM column in case the player didn't play. We can remove the rest of the columns.

```
In [19]: def get_minutes_stats(df):
        """For a given dataframe, retains the Player Name, FGM, Min, Starter and GameID
        df = df[['Player Name','FGM','Min','Starter','GameID']]
        return df

        miami_boxscores = get_minutes_stats(miami_boxscores)
        miami_boxscores.head()
```

Out[19]:

	Player Name	FGM	Min	Starter	GameID
351	Josh Richardson	2	16:04	1	0021701004
352	James Johnson	3	24:02	1	0021701004
353	Bam Adebayo	1	24:40	1	0021701004
354	Tyler Johnson	6	34:48	1	0021701004
355	Goran Dragic	10	30:45	1	0021701004

At the moment, we can't be sure that the game ordering is the same as it is in the results dataframe (which is sorted by date). A solution to this is to 'look-up' the boxscore GameID's in the results dataframe and return the date for that GameID. This can be accomplished by using a left inner join between the boxscore dataframe and the GameID and GameDate columns of the results dataframe.

```
In [20]: def get_boxscore_dates(df):
        """Takes in a dataframe with a GameID column and uses a left inner join with the
        corresponding GameDate"""
        df = pd.merge(df, results_data[['GameID','GameDate']], on = 'GameID')
        # join on GameID as that's the common column between the 2 dataframes
        df.sort_values(by='GameDate', inplace=True)
        # sort by date and return
        return df

        miami_boxscores = get_boxscore_dates(miami_boxscores)
        miami_boxscores.head()
```

Out[20]:

	Player Name	FGM	Min	Starter	GameID	GameDate
848	Goran Dragic	6	30:37	1	0021700005	2017-10-18 00:00:00
856	Matt Williams Jr.	DNP - Coach's Decision		0	0021700005	2017-10-18 00:00:00
855	Jordan Mickey	DNP - Coach's Decision		0	0021700005	2017-10-18 00:00:00
854	Udonis Haslem	DNP - Coach's Decision		0	0021700005	2017-10-18 00:00:00
853	Wayne Ellington	1	9:57	0	0021700005	2017-10-18 00:00:00

We can see that we have a date column now, but we need to deal with the blank 'Min' values which have an empty string in them (so we can't use fillna) and move the reason for not playing into a new column, replacing the FGM with zero (if a player didn't play then they didn't score any field goals).

```
In [21]: def dnp_reason(row):
        """Takes in a row of a dataframe, creating a new column called 'DNP Reason' wh
        if the player played and otherwise contains a reason for the player not playin
        play then their minutes and FGM columns are set to zero"""
        reason = row['FGM']
        try :
            # if the reason column casts as an int, then the player must have played
            reason = int(reason)
            # players that played don't have a DNP reason
            row['DNP Reason'] = 0
        except ValueError:
            # if it can't be cast as an int then it must be a string
            row['FGM'] = 0 # put a zero in the FGM column
            row['Min'] = '0:0' # keep minutes format same as other rows for now
            # splitting the string by spaces returns the reason in index 2
            if reason == '':
                row['DNP Reason'] = 0
            elif reason.split(' ')[2] == 'Injury/Illness':
                row['DNP Reason'] = 'Injury/Illness'
            elif reason.split(' ')[2] == "Coach's":
                row['DNP Reason'] = 0
            else:
                row['DNP Reason'] = 'Other'
        return row

        miami_boxscores = miami_boxscores.apply(dnp_reason, axis=1)
        miami_boxscores.head()
```

Out[21]:

	Player Name	FGM	Min	Starter	GameID	GameDate	DNP Reason
848	Goran Dragic	6	30:37	1	0021700005	2017-10-18 00:00:00	
856	Matt Williams Jr.	0	0:0	0	0021700005	2017-10-18 00:00:00	
855	Jordan Mickey	0	0:0	0	0021700005	2017-10-18 00:00:00	
854	Udonis Haslem	0	0:0	0	0021700005	2017-10-18 00:00:00	
853	Wayne Ellington	1	9:57	0	0021700005	2017-10-18 00:00:00	

Quick check to see if we have any mis-spellings of player names that cause duplicates.

```
In [22]: miami_players = miami_boxscores['Player Name'].unique()
        miami_players
```

```
Out[22]: array(['Goran Dragic', 'Matt Williams Jr.', 'Jordan Mickey',
        'Udonis Haslem', 'Wayne Ellington', 'Bam Adebayo',
        'Justise Winslow', 'James Johnson', 'Tyler Johnson',
        'Dion Waiters', 'Hassan Whiteside', 'Kelly Olynyk',
        'Josh Richardson', 'Okaro White', 'Rodney McGruder',
        'Derrick Walton Jr.', 'AJ Hammons', 'Derrick Jones Jr.',
        'Dwyane Wade', 'Luke Babbitt'], dtype=object)
```

Looks good, each name only appears once. Now we need to convert the minutes column from a string with format 'mm:ss' to a decimal for use in a heatmap.


```
In [23]: def convert_mins_decimal(row):
        """Takes in a dataframe row with a 'Min' column of format 'mm:ss' and converts
        min_string = row['Min']
        minutes = float(min_string.split(':')[0]) # minutes contained in string before
        seconds = float(min_string.split(':')[1])/60 # seconds contained after the ':'
        row['Min'] = round(minutes+seconds, 2) # add together and round to 2 dp
        return row

        miami_boxscores = miami_boxscores.apply(convert_mins_decimal, axis=1)
        miami_boxscores.head()
```

Out[23]:

	Player Name	FGM	Min	Starter	GameID	GameDate	DNP Reason
848	Goran Dragic	6	30.62	1	0021700005	2017-10-18 00:00:00	
856	Matt Williams Jr.	0	0.00	0	0021700005	2017-10-18 00:00:00	
855	Jordan Mickey	0	0.00	0	0021700005	2017-10-18 00:00:00	
854	Udonis Haslem	0	0.00	0	0021700005	2017-10-18 00:00:00	
853	Wayne Ellington	1	9.95	0	0021700005	2017-10-18 00:00:00	

Currently, each player has as many rows as the number of games they played. Plotting this as a heatmap would be very narrow but very long. Instead, we would like 1 row per player with 1 column per game. To do this we can create a new dataframe which is grouped by Player Name. We would like the players in order of total minutes played in the season.

```
In [24]: def create_plotting_df(df):
        """Takes a teams boxscore dataframe and groups by player name, summing the min
        players by total minutes played in the season."""
        plotting_df = df.groupby(by='Player Name').sum()[['Min']].sort_values(by='Min')
        plotting_df.reset_index(inplace=True) # reset index so we get integer indexes

        return plotting_df

        miami_plotting_df = create_plotting_df(miami_boxscores)
        miami_plotting_df.head()
```

Out[24]:

	Player Name	Min
0	Josh Richardson	2262.50
1	Goran Dragic	2029.60
2	Tyler Johnson	1715.87
3	Wayne Ellington	1693.75
4	James Johnson	1588.22

Now we need to build up the columns. We could use the GameDates as the columns, but then plotting them on an axis would lead to uneven spacing due to the differing numbers of days between games. Instead we use GameNumber (like we did for the results data plot). This time, we can't simply use the index to create this because there are multiple players per game. One way to do it is to build up a dictionary by counting the number of unique dates and assigning the count as their game number.

```
In [25]: def get_boxscore_gamenum(df):
        """Takes in a dataframe with a date column and assigns an integer game number
        game_dates = df['GameDate'].unique()
        date_dict = {}
        count=1 # count begins at 1 because the first date is the date of the first ga
        for date in game_dates:
            date_dict[date] = count # create a mapping 'date':GameNumber
            count +=1 # increment count and move to next date

        df['GameNumber'] = df['GameDate'].map(date_dict)
        # create a new column which is the GameDate column mapped using the dictionary

        return df

        miami_boxscores = get_boxscore_gamenum(miami_boxscores)
        miami_boxscores.head()
```

Out[25]:

	Player Name	FGM	Min	Starter	GameID	GameDate	DNP Reason	GameNumber
848	Goran Dragic	6	30.62	1	0021700005	2017-10-18 00:00:00		1
856	Matt Williams Jr.	0	0.00	0	0021700005	2017-10-18 00:00:00		1
855	Jordan Mickey	0	0.00	0	0021700005	2017-10-18 00:00:00		1
854	Udonis Haslem	0	0.00	0	0021700005	2017-10-18 00:00:00		1
853	Wayne Ellington	1	9.95	0	0021700005	2017-10-18 00:00:00		1

Create a blank column in plotting_df for every game

```
In [26]: def create_blanks_plotting(plotting_df, boxscores_df):
        """Takes in a plotting df which consists of a teams player name and total minu
        then adds a column of zeros per game"""
        for num in boxscores_df['GameNumber'].unique():
            plotting_df[num] = 0

        return plotting_df

        miami_plotting_df = create_blanks_plotting(miami_plotting_df, miami_boxscores)
        miami_plotting_df.head()
```

Out[26]:

	Player Name	Min	1	2	3	4	5	6	7	8	...	60	61	62	63	64	65	66	67	68	69
0	Josh Richardson	22	6	2	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	Goran Dragic	20	2	9	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Tyler Johnson	17	1	5	8	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Wayne Ellington	16	9	3	7	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	James Johnson	15	8	8	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 71 columns

Now for each Player Name-GameNumber pair we can look up the minutes played in the boxscore dataframe and insert it into the correct place in the plotting dataframe. Some of the pairs won't appear in the boxscore dataframe because some players won't appear in some boxscores due long-term injuries, player trades etc. These will be set to zero minutes and we will deal with them later.

```
In [27]: def get_minutes_dates(row, boxscores_df):
        """Takes in a row containing a player name for a team and returns columns containing
        minutes played for each game the team has played. If the player doesn't appear
        their minutes are set to zero"""

        player = row['Player Name']
        for num in boxscores_df['GameNumber'].unique():
            mins = boxscores_df[(boxscores_df['Player Name'] == player) & (boxscores_df['GameNumber'] == num)]
            # mins contains an empty list if the player doesn't appear in the boxscore
            if len(mins.values) == 0 :
                row[num] = 0
            else:
                # if the player does appear then take the minutes value
                row[num] = mins.values[0]
        return row

        miami_plotting_df = miami_plotting_df.apply(get_minutes_dates, args=(miami_boxscore_df,))
        miami_plotting_df.head()
```

Out[27]:

	Player Name	Min	1	2	3	4	5	6	7	8...	60	61	62	63	64	65	
0	Josh Richardson	2262.50	34.93	37.57	36.88	33.00	35.83	32.42	22.20	32.63	...	27.00	37.82	35.53	35.40	18.70	35.07
1	Goran Dragic	2029.60	30.62	32.15	32.58	37.38	36.40	39.12	34.38	34.23	...	25.23	33.40	34.72	31.00	23.45	35.62
2	Tyler Johnson	1715.87	21.30	26.28	26.85	33.27	22.80	22.83	31.62	25.55	...	31.55	22.38	0.00	0.00	18.07	38.23
3	Wayne Ellington	1693.75	9.95	15.40	24.83	6.30	17.07	3.87	23.15	12.45	...	27.97	11.45	0.00	0.00	0.00	0.00
4	James Johnson	1588.22	18.70	29.73	26.42	36.02	32.63	23.67	0.00	30.40	...	19.38	16.43	25.07	26.75	20.88	14.37

5 rows × 71 columns

3. Flexible Data Visualisation

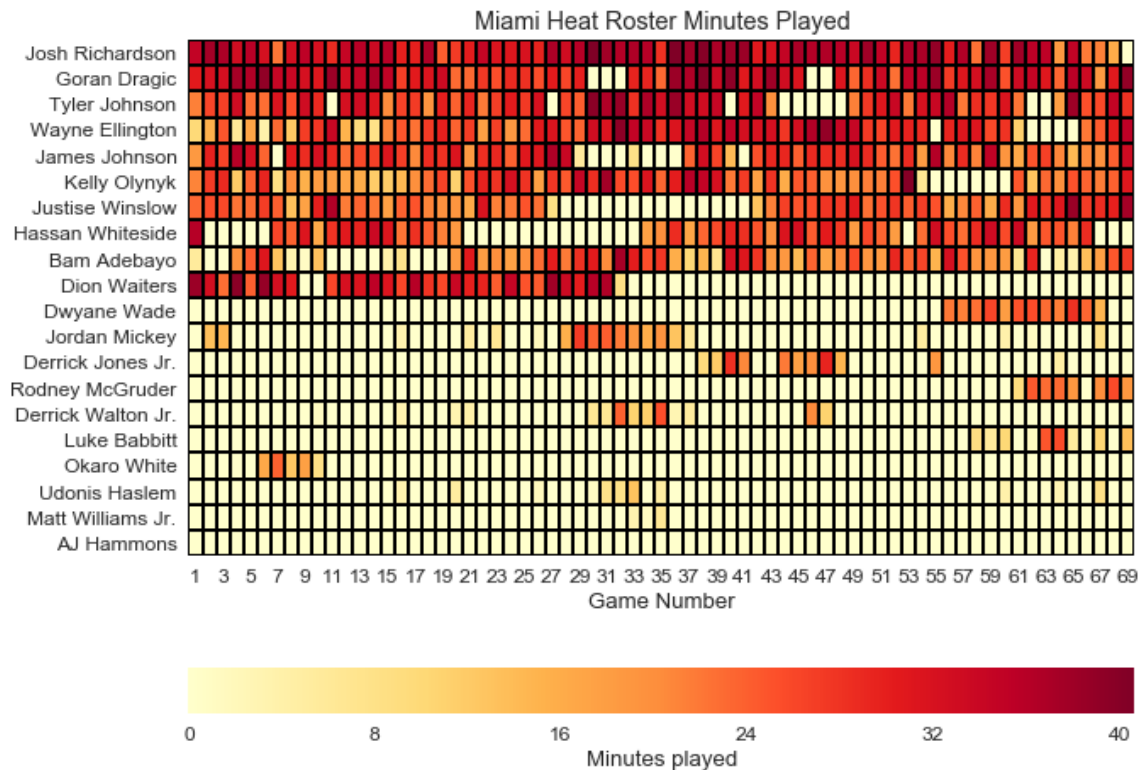
Going back to the original aim of this notebook, we would like to create a visualisation that displays a number of different pieces of information at once.

To start with we can use the Seaborn library to make a simple heatmap of the minutes played per game.

```
In [28]: import seaborn as sns
sns.set(font_scale=1.2) # set a larger font scale to increase general font size
fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on
sns.heatmap(miami_plotting_df.drop(['Player Name', 'Min'], axis=1), xticklabels=2,
            yticklabels=miami_plotting_df['Player Name'], linewidths=1, linecolor=
            cbar_kws={'label': 'Minutes played', 'orientation': 'horizontal'}, cmap=

plt.title('Miami Heat Roster Minutes Played')
plt.xlabel('Game Number')
```

```
Out[28]: Text(0.5,181.464,'Game Number')
```



This looks good, but doesn't give all of the information I set out to convey. We can't tell the difference between injured players and players not playing for other reason, and for example Dwayne Wade joined the team in a trade around game 55, so any game before that he was playing for a different team. We also can't tell how well the team is doing throughout the season, or who was in the starting lineup for a particular game.

Adding custom annotations

One way to add additional information to the heatmap is to add annotations. The default annotation is the value of the cell in the heatmap which wouldn't be useful for us. Instead, we can supply a dataframe with identical dimensions to the plotted dataframe with the actual annotations that we would like.

```

In [29]: annot_df = miami_plotting_df.copy()
def get_annots(row, boxscores_df):
    """Takes in a row with a player name and for each game, checks if the player w
    A DNP reason of Injury/Illness will be represented with a '/', and a starter w
    player = row['Player Name']
    for num in boxscores_df['GameNumber'].unique():
        starter = boxscores_df[(boxscores_df['Player Name'] == player) & (boxscore
        dnp = boxscores_df[(boxscores_df['Player Name'] == player) & (boxscores_df
        if len(dnp) == 0: # this player wasn't in the boxscore
            row[num] = '' # no annotation
        elif starter.values[0] == 1: # this player was a starter
            row[num] = '=' # insert annotation
        elif dnp.values[0] == 'Injury/Illness': # this player was injured
            row[num] = '/' # insert annotation
        else:
            row[num] = '' # this player did not start and was not injured
        pass

    return row

annot_df = annot_df.apply(get_annots, args=(miami_boxscores,), axis=1)
annot_df.head()

```

Out[29]:

	Player Name	Min	1	2	3	4	5	6	7	8	...	60	61	62	63	64	65	66	67	68	69
0	Josh Richardson	22	6	5	0	=	=	=	=	=	...	=	=	=	=	=	=	=	=	=	/
1	Goran Dragic	20	2	9	6	0	=	=	=	=	...	=	=	=	=	=	=	=	=	=	=
2	Tyler Johnson	17	1	5	8	7					...	=	=					=	=	=	=
3	Wayne Ellington	16	9	3	7	5					...										
4	James Johnson	15	8	8	2	2	=	=	=	=	/	...			=	=	=	=	=	=	=

5 rows × 71 columns

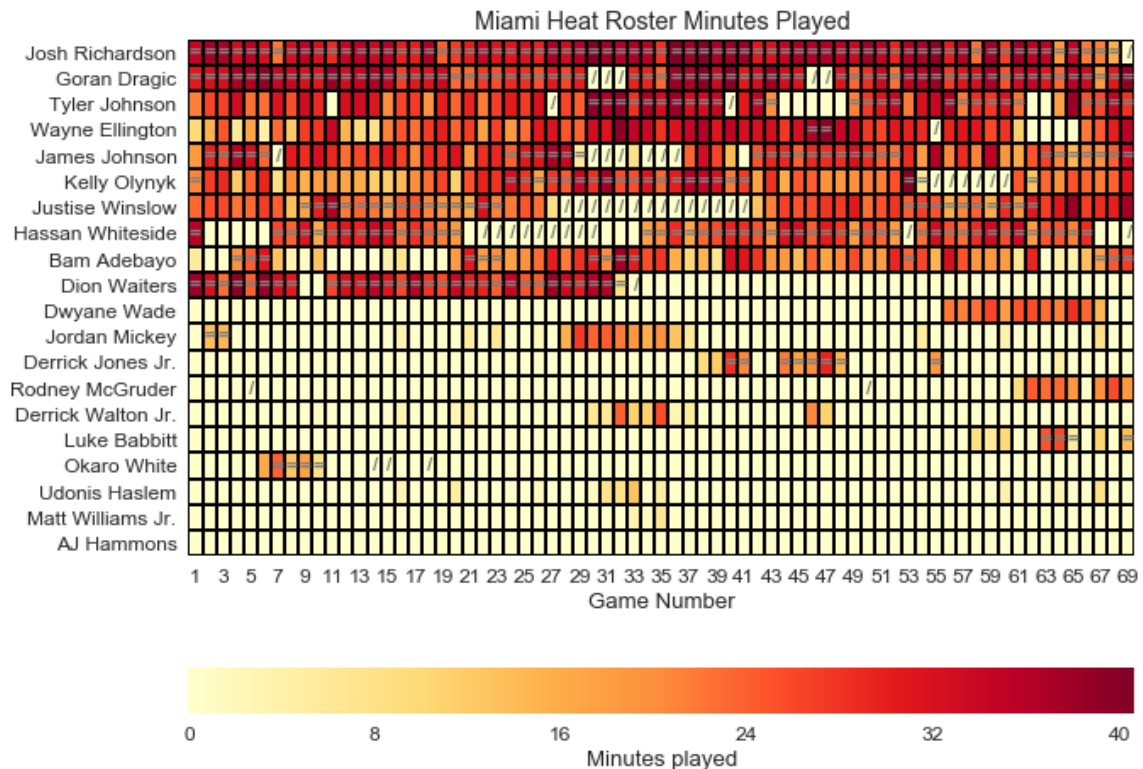
```
In [30]: # Let's try plotting again
fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on

annot_kws = {"size":12, "color":"grey", "weight":"bold"} # formatting choices for

sns.heatmap(miami_plotting_df.drop(['Player Name','Min'], axis=1),xticklabels=2,
            yticklabels=miami_plotting_df['Player Name'], linewidths=1, linecolor=
            cbar_kws={'label':'Minutes played', 'orientation':'horizontal'}, cmap=
            annot=annot_df.drop(['Player Name','Min'], axis=1),fmt='', annot_kws=an

plt.title('Miami Heat Roster Minutes Played')
plt.xlabel('Game Number')
```

```
Out[30]: Text(0.5,181.464,'Game Number')
```



We can now distinguish players that started games and injured players from the rest.

Masking Heatmaps

The next step is to represent players that weren't in the boxscore at all by greying out those boxes. We do this by first creating a 'masking' dataframe which contains a 1 where we would like to grey out and a zero everywhere else.

```
In [31]: def get_nwt(row, boxscores_df):
        """Given a row with a player name in, returns a 1 in games where the player was
        if they were in the box score"""
        player = row['Player Name']
        for num in boxscores_df['GameNumber'].unique():
            boxscore_row = boxscores_df[(boxscores_df['Player Name'] == player) & (box
            if len(boxscore_row) == 0:
                row[num] = 1
            else:
                row[num] = 0
        return row

        mask_df = annot_df.apply(get_nwt, args=(miami_boxscores,), axis=1)
        mask_df.head()
```

Out[31]:

	Player Name	Min	1	2	3	4	5	6	7	8	...	60	61	62	63	64	65	66	67	68	69
0	Josh Richardson	22	62.50	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Goran Dragic	20	29.60	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	Tyler Johnson	17	15.87	0	0	0	0	0	0	0	...	0	0	1	1	0	0	0	0	0	0
3	Wayne Ellington	16	93.75	0	0	0	0	0	0	0	...	0	0	1	1	1	1	0	0	0	0
4	James Johnson	15	88.22	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 71 columns

We can now use this dataframe to plot a solid color when the mask is equal to 1 and nothing otherwise.

```

In [32]: # Let's try plotting again
fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on

annot_kws = {"size":12, "color":"grey", "weight":"bold"} # formatting choices for

# first plot our actual heatmap
sns.heatmap(miami_plotting_df.drop(['Player Name','Min'], axis=1),xticklabels=2,
            yticklabels=miami_plotting_df['Player Name'], linewidths=1, linecolor=
            cbar_kws={'label':'Minutes played', 'orientation':'horizontal'}, cmap=
            annot=annot_df.drop(['Player Name','Min'], axis=1),fmt='', annot_kws=an

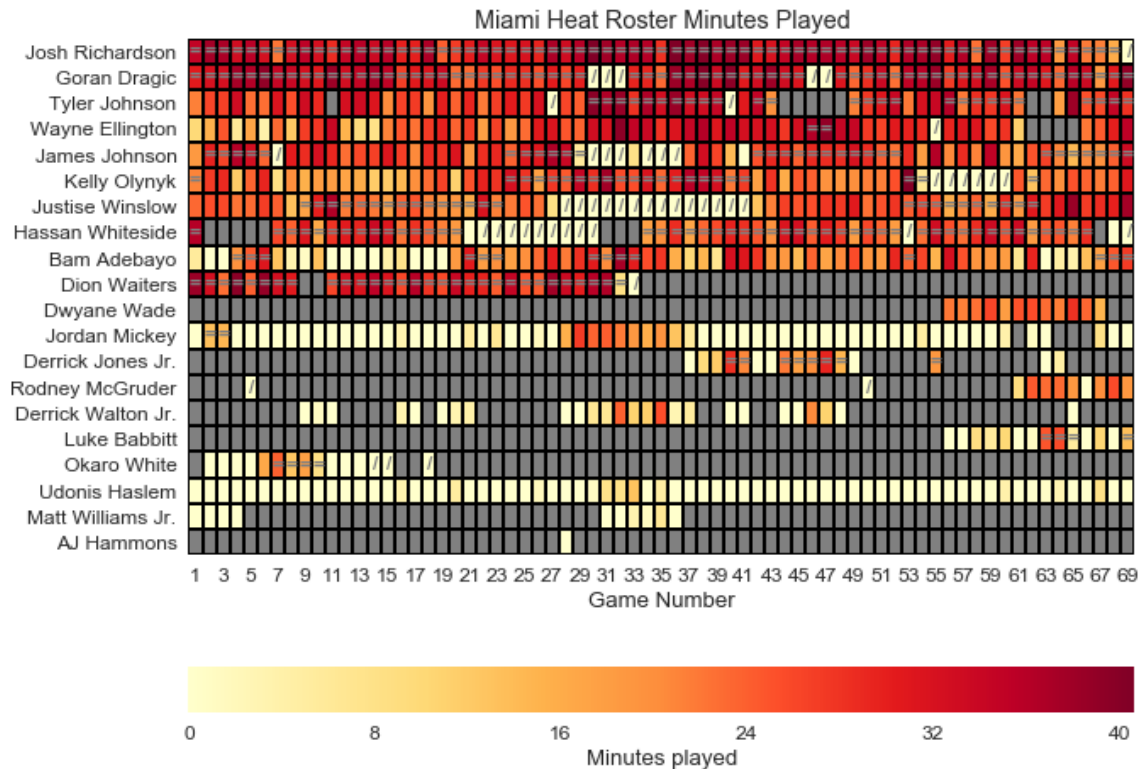
# import ListedColormap so that we can create a specific color for a specific valu
from matplotlib.colors import ListedColormap

# now plot the masked dataframe, specifying the colors for value 0 and 1, and spec
sns.heatmap(mask_df.drop(['Player Name','Min'], axis=1),
            mask=mask_df.drop(['Player Name','Min'], axis=1)<1,
            cmap=ListedColormap(['grey', 'grey']), linewidths=1,
            linecolor='black', cbar=False ,yticklabels=miami_plotting_df['Player N

plt.title('Miami Heat Roster Minutes Played')
plt.xlabel('Game Number')

```

Out[32]: Text(0.5,181.464,'Game Number')



This is looking much better and conveying much more information. The only things left to add are the season progress and a legend for the annotations and grey blocks.

Overlaying a lineplot and custom legends

To overlay the lineplot we can simply create a copy of the x-axis used for the heatmap (using the `twinx()` function) and plot ontop of this as normal. We then need to align the y-ticks based on the required range of y-values and the existing gridlines from the heatmap.

```

In [33]: # Let's try plotting again
fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on
ax=fig.add_subplot(111, label="1") # define an axis object so we can copy it

annot_kws = {"size":12, "color":"grey", "weight":"bold"} # formatting choices for

# first plot our actual heatmap
sns.heatmap(miami_plotting_df.drop(['Player Name','Min'], axis=1),xticklabels=2, a
            yticklabels=miami_plotting_df['Player Name'], linewidths=1, linecolor=
            cbar_kws={'label':'Minutes played', 'orientation':'horizontal'}, cmap=
            annot=annot_df.drop(['Player Name','Min'], axis=1),fmt='', annot_kws=an

# import ListedColormap so that we can create a specific color for a specific valu
from matplotlib.colors import ListedColormap

# now plot the masked dataframe, specifying the colors for value 0 and 1, and spec
sns.heatmap(mask_df.drop(['Player Name','Min'], axis=1), ax=ax,
            mask=mask_df.drop(['Player Name','Min'], axis=1)<1,
            cmap=ListedColormap(['grey', 'grey']), linewidths=1,
            linecolor='black', cbar=False ,yticklabels=miami_plotting_df['Player N

# now copy the axis and plot the line plot
ax2=ax.twinx() # copy axis
ax2.axhline(y=0, color='black', linewidth=10, zorder=-1) # create a horizontal lin
ax2.plot([0] + list(miami_games['GameNumber']), [0] + list(miami_games['+/-']),
        drawstyle='steps-post', linewidth=6, zorder=1) # plot the line
ax2.set_ylabel('Wins minus Losses') # define the y label
ax2.yaxis.tick_right() # set the ticks to appear on the RHS
ax2.yaxis.set_label_position('right') # set the label to appear on the right

# code to set the yticks for the line plot
from math import gcd
# find the maximum value so we can set the yrange relative to that
absmaxval = miami_games['+/-'].abs().max()
# if the value is even, add 2 to it so we have an even range above and below zero
if absmaxval % 2 == 0:
    hcf = gcd(2*(absmaxval + 2), len(list(miami_boxscores['Player Name'].unique()))
    # use the highest common factor plus one to make sure we are plotting integers
    # this only works if there are an even number of players
    # need to pad for teams that have an odd number of players
    ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 2), (absmaxval + 2), num=hcf+1

else:
    # if the value is odd, add 1 to make it even
    hcf = gcd(2*(absmaxval + 1), len(list(miami_boxscores['Player Name'].unique()))
    # use the highest common factor plus one to make sure we are plotting integers
    # this only works if there are an even number of players
    # need to pad for teams that have an odd number of players
    ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 1), (absmaxval + 1), num=hcf+1

ax2.grid(False) # don't display the grid for the lineplot

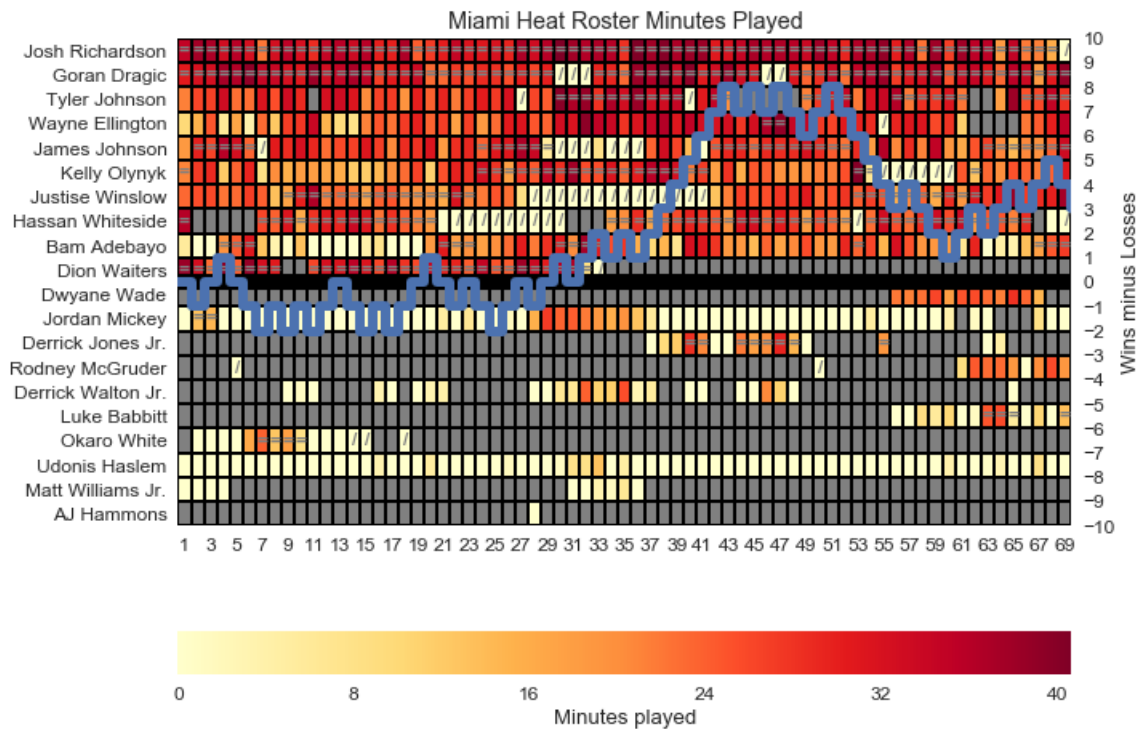
plt.title('Miami Heat Roster Minutes Played')
plt.xlabel('Game Number')

```

```

Out[33]: Text(0.5,0,'Game Number')

```



The last piece of the puzzle is adding the legend. Seeing as the elements we need a legend for all come from different plots we can't use the automatic legend. What we can do is create 'proxy' lines, which do not appear on the chart but have a label equal to the whole string we want to display in the legend. And for the greyed out cells we can simply add a grey patch to the legend.

```
In [34]: labels = ['=', '/'] # the markers used for the 2 annotations
descriptions = ['Starter', 'Injured/Ill', 'Not in boxscore', 'Win/Loss'] # the desc
# note 'Not in boxscore' and 'Win/Loss' in the descriptions list but not the label
# this is because we will add the label manually
```

```
In [35]: import matplotlib.lines

# function that creates the fake blank lines
def create_proxy(label):
    line = matplotlib.lines.Line2D([0], [0], linestyle='none', mfc='black',
                                    mec='none', marker=r'$\mathregular{{{}}}$'.format(label))
    return line
proxies = [create_proxy(item) for item in labels]
```

In [36]:

```

# Let's try plotting again
fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on
ax=fig.add_subplot(111, label="1") # define an axis object so we can copy it

annot_kws = {"size":12, "color":"grey", "weight":"bold"} # formatting choices for

# first plot our actual heatmap
sns.heatmap(miami_plotting_df.drop(['Player Name','Min'], axis=1),xticklabels=2, a
yticklabels=miami_plotting_df['Player Name'], linewidths=1, linecolor=
cbar_kws={'label':'Minutes played', 'orientation':'horizontal'}, cmap=
annot=annot_df.drop(['Player Name','Min'], axis=1),fmt='', annot_kws=an

# now plot the masked dataframe, specifying the colors for value 0 and 1, and spec
sns.heatmap(mask_df.drop(['Player Name','Min'], axis=1), ax=ax,
mask=mask_df.drop(['Player Name','Min'], axis=1)<1,
cmap=ListedColormap(['grey', 'grey']), linewidths=1,
linecolor='black', cbar=False ,yticklabels=miami_plotting_df['Player N

# now copy the axis and plot the line plot
ax2=ax.twinx() # copy axis
ax2.axhline(y=0, color='black', linewidth=10, zorder=-1) # create a horizontal lin
ax2.plot([0] + list(miami_games['GameNumber']), [0] + list(miami_games['+/-']),
drawstyle='steps-post', linewidth=6, zorder=1) # plot the line
ax2.set_ylabel('Wins minus Losses') # define the y label
ax2.yaxis.tick_right() # set the ticks to appear on the RHS
ax2.yaxis.set_label_position('right') # set the label to appear on the right

# code to set the yticks for the line plot
from math import gcd
# find the maximum value so we can set the yrange relative to that
absmaxval = miami_games['+/-'].abs().max()
# if the value is even, add 2 to it so we have an even range above and below zero
if absmaxval % 2 == 0:
    hcf = gcd(2*(absmaxval + 2), len(list(miami_boxscores['Player Name'].unique()))
    # use the highest common factor plus one to make sure we are plotting integers
    # this only works if there are an even number of players
    # need to pad for teams that have an odd number of players
    ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 2), (absmaxval + 2), num=hcf+1

else:
    # if the value is odd, add 1 to make it even
    hcf = gcd(2*(absmaxval + 1), len(list(miami_boxscores['Player Name'].unique()))
    # use the highest common factor plus one to make sure we are plotting integers
    # this only works if there are an even number of players
    # need to pad for teams that have an odd number of players
    ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 1), (absmaxval + 1), num=hcf+1
ax2.grid(False) # don't display the grid for the lineplot

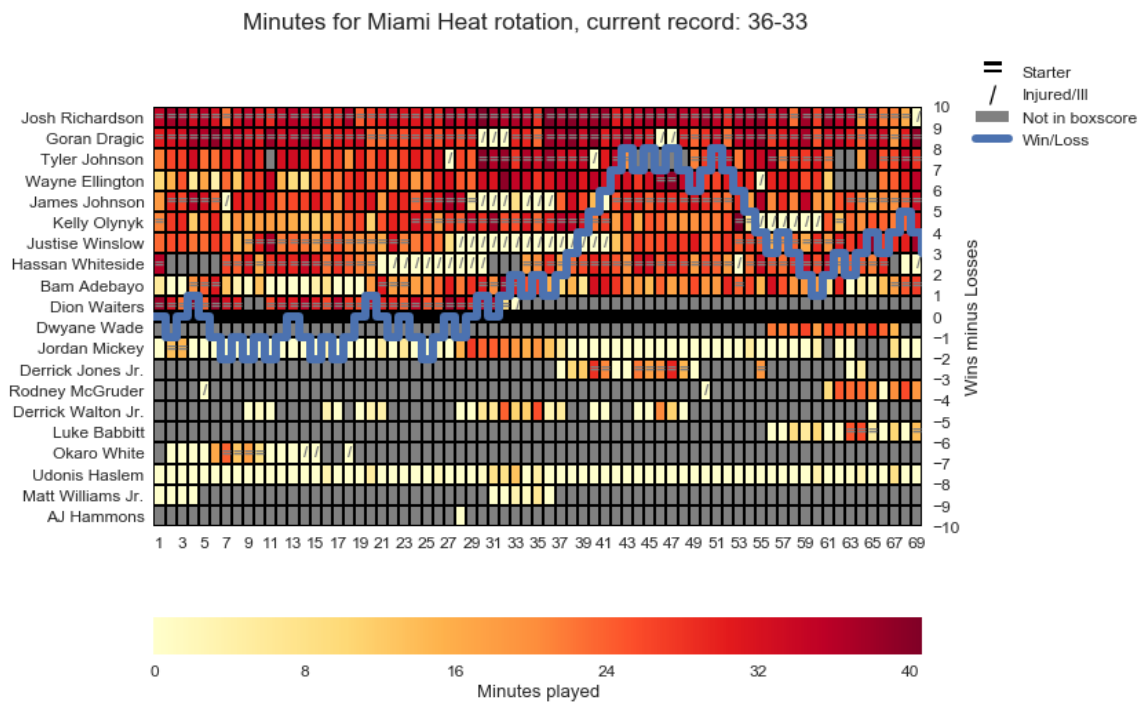
# new code for the custom legend
import matplotlib.patches as patches # import patches so we can create a rectangle
line = matplotlib.lines.Line2D([0], [0], mfc='blue', linewidth=6)
ax2.legend(proxies + [patches.Rectangle((0,0),1,1,facecolor='grey')]) + [line], des
markerscale=2, loc='center left', bbox_to_anchor=(1.05, 1))

# custom title that includes wins and losses
losses = (len(list(miami_games['GameNumber'])) - list(miami_games['+/-'])[-1])/2
wins = losses + list(miami_games['+/-'])[-1]
fig.suptitle('Minutes for Miami Heat rotation, current record: ' + str(int(wins))

plt.xlabel('Game Number')

```

```
Out[36]: █ Text(0.5,0,'Game Number')
```



Now we have our final plot, which shows a large amount of information in a relatively compact space. There may be some ways to make the plot more visually pleasing but I'm more interested in the process than making the final plot perfect. Like with the results data plot, it would be great to be able to create a pipeline that allows us to plot this graph with just a single line of code for each team.

In [37]:

```

# first turn the plot code above into a function
def plot_heatmap(games_df, boxscores_df, plotting_df, annot_df, mask_df, team_name):
    """Takes in the results dataframe for a particular team, the boxscores df for
    the annotation dataframe, plotting dataframe and the mask dataframe,
    along with the string of the team name, and plots the complete heatmap"""

    fig = plt.figure(figsize=(10,8)) # define the size of the canvas to plot on
    ax=fig.add_subplot(111, label="1") # define an axis object so we can copy it

    annot_kws = {"size":12, "color":"grey", "weight":"bold"} # formatting choices

    # first plot our actual heatmap
    sns.heatmap(plotting_df.drop(['Player Name','Min'], axis=1),xticklabels=2, ax=
                yticklabels=plotting_df['Player Name'], linewidths=1, linecolor='b
                cbar_kws={'label':'Minutes played', 'orientation':'horizontal'}, c
                annot=annot_df.drop(['Player Name','Min'], axis=1),fmt='', annot_kw

    # now plot the masked dataframe, specifying the colors for value 0 and 1, and
    sns.heatmap(mask_df.drop(['Player Name','Min'], axis=1), ax=ax,
                mask=mask_df.drop(['Player Name','Min'], axis=1)<1,
                cmap=ListedColormap(['grey', 'grey']), linewidths=1,
                linecolor='black', cbar=False ,yticklabels=plotting_df['Player Nam

    # now copy the axis and plot the line plot
    ax2=ax.twinx() # copy axis
    ax2.axhline(y=0, color='black', linewidth=10, zorder=-1) # create a horizontal
    ax2.plot([0] + list(games_df['GameNumber']), [0] + list(games_df['+/-']),
             drawstyle='steps-post', linewidth=6, zorder=1) # plot the line
    ax2.set_ylabel('Wins minus Losses') # define the y label
    ax2.yaxis.tick_right() # set the ticks to appear on the RHS
    ax2.yaxis.set_label_position('right') # set the label to appear on the right

    # code to set the yticks for the line plot
    from math import gcd
    # find the maximum value so we can set the yrange relative to that
    absmaxval = games_df['+/-'].abs().max()
    # if the value is even, add 2 to it so we have an even range above and below z
    if absmaxval % 2 == 0:
        hcf = gcd(2*(absmaxval + 2), len(list(plotting_df['Player Name'].unique()))
        # use the highest common factor plus one to make sure we are plotting inte
        # this only works if there are an even number of players
        # need to pad for teams that have an odd number of players
        ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 2), (absmaxval + 2), num=h

    else:
        # if the value is odd, add 3 to make it even
        hcf = gcd(2*(absmaxval + 3), len(list(plotting_df['Player Name'].unique()))
        # use the highest common factor plus one to make sure we are plotting inte
        # this only works if there are an even number of players
        # need to pad for teams that have an odd number of players
        ax2.yaxis.set_ticks(np.linspace(-1*(absmaxval + 3), (absmaxval + 3), num=h
    ax2.grid(False) # don't display the grid for the lineplot

    # code for the custom legend
    import matplotlib.lines
    labels = ['=', '/'] # the markers used for the 2 annotations
    descriptions = ['Starter', 'Injured/Ill', 'Not in boxscore', 'Win/Loss']

    proxies = [create_proxy(item) for item in labels]

    import matplotlib.patches as patches # import patches so we can create a recta

```

```

line = matplotlib.lines.Line2D([0], [0], mfc='blue', linewidth=6)
ax2.legend(proxies + [patches.Rectangle((0,0),1,1,facecolor='grey')] + [line],
           markerscale=2, loc='center left', bbox_to_anchor=(1.05, 1))

# custom title that includes wins and losses
losses = (len(list(games_df['GameNumber'])) - list(games_df['+/-'])[-1])/2
wins = losses + list(games_df['+/-'])[-1]
fig.suptitle('Minutes for ' + team_name + ' rotation, current record: ' + str(
plt.xlabel('Game Number'))

```

```

In [38]: # modify our original pipeline to remove the plotting part
def results_pipeline_noplot(team_name):
    """Takes in a string of the team name (case sensitive) and processes it for th
    """
    # First check if the team name is valid
    if team_name in team_names:
        # then call the sequence of functions defined above
        df = get_team_games(team_name)
        df = df.apply(calc_wins, args=(team_name,), axis=1)
        df = add_plus_minus(df)
        df = add_game_number(df)
        # plus_minus_plot(df, team_name) remove the first basic plot from the func
    return df

```

```

In [39]: # create a new pipeline
def heatmap_pipeline(team_name):
    """This function takes in a string of a team name and plots the complete heatm
    for that team."""
    if team_name not in team_names:
        print('Invalid team name')
    else:
        games_df = results_pipeline_noplot(team_name)
        boxscores_df = get_team_boxscores(team_name)
        boxscores_df = get_minutes_stats(boxscores_df)
        boxscores_df = get_boxscore_dates(boxscores_df)
        boxscores_df = boxscores_df.apply(dnp_reason, axis=1)
        boxscores_df = boxscores_df.apply(convert_mins_decimal, axis=1)
        plotting_df = create_plotting_df(boxscores_df)

        # if there's an odd number of players then add a blank row so that our plo
        if len(plotting_df['Player Name'].unique()) % 2 != 0:
            plotting_df = plotting_df.append({'Player Name': ' ', 'Min':0}, ignore_
        else:
            pass

        boxscores_df = get_boxscore_gamenum(boxscores_df)
        plotting_df = create_blanks_plotting(plotting_df, boxscores_df)
        plotting_df = plotting_df.apply(get_minutes_dates, args=(boxscores_df,), ax
        annot_df = plotting_df.apply(get_annots, args=(boxscores_df,), axis=1)
        mask_df = annot_df.apply(get_nwt, args=(boxscores_df,), axis=1)

        plot_heatmap(games_df, boxscores_df, plotting_df, annot_df, mask_df, team_




```

```
In [40]: # using the following code, we could very easily plot the graphs for every team  
# for team_name in team_names:  
    # heatmap_pipeline(team_name)
```

This concludes our data collection, cleaning and visualisation exercise. The functions above are not optimal because I want to show a logical progression through the notebook. If you would like to see my 'in-use' versions of the above functions please look at the 'dataviz.py' script file in the github repository ([here \(https://github.com/HexhamAllstar/NBA-Scraper\)](https://github.com/HexhamAllstar/NBA-Scraper)).

Every day (or every day that there are matches) I can run 2 simple functions from an interactive python console and my local database will be updated and the graphs above will be regenerated and saved.

Thanks for reading!

Tags  Python (<https://hexhamallstar.github.io/tag/python.html>)  Selenium (<https://hexhamallstar.github.io/tag/selenium.html>)  Webscraping (<https://hexhamallstar.github.io/tag/webscraping.html>) 
Data Visualisation (<https://hexhamallstar.github.io/tag/data-visualisation.html>)  SQL (<https://hexhamallstar.github.io/tag/sql.html>)  Basketball (<https://hexhamallstar.github.io/tag/basketball.html>)  Data Cleaning (<https://hexhamallstar.github.io/tag/data-cleaning.html>)

Like 0 Share

Comments

hexhamallstar Comment Policy

Feel free to leave me any feedback on the content of my posts!



0 Comments

hexhamallstar

3 Paul DeVos ▾

♥ Recommend

🐦 Tweet

📱 Share

Sort by Best ▾



Start the discussion...

Be the first to comment.

✉ Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

💬 Social

LinkedIn

(<http://www.linkedin.com/in/philbingham45/>)

GitHub (<http://github.com/hexhamallstar>)

📁 Categories

Notebooks

(<https://hexhamallstar.github.io/category/notebook>)

Tableau

(<https://hexhamallstar.github.io/category/tableau>)

🔗 Links

r/datascience (<https://www.reddit.com/r/datascience/>)

r/machinelearning (<https://www.reddit.com/r/MachineLearning/>)

NBA (<https://www.nba.com>)

DataScienceCentral (<https://www.datasciencecentral.com/>)

KDNuggets (<https://www.kdnuggets.com/>)

🏷️ Tags

Tableau (<https://hexhamallstar.github.io/tag/tableau.html>)
 Dashboard (<https://hexhamallstar.github.io/tag/dashboard.html>)
 NBA (<https://hexhamallstar.github.io/tag/nba.html>)
 Basketball (<https://hexhamallstar.github.io/tag/basketball.html>)
 SQL (<https://hexhamallstar.github.io/tag/sql.html>)
 Python (<https://hexhamallstar.github.io/tag/python.html>)
 Recommender System (<https://hexhamallstar.github.io/tag/recommender-system.html>)
 MovieLens (<https://hexhamallstar.github.io/tag/movielens.html>)
 PySpark (<https://hexhamallstar.github.io/tag/pyspark.html>)
 Spark (<https://hexhamallstar.github.io/tag/spark.html>)
 ALS (<https://hexhamallstar.github.io/tag/als.html>)
 Keras (<https://hexhamallstar.github.io/tag/keras.html>)
 CNN (<https://hexhamallstar.github.io/tag/cnn.html>)
 Supervised (<https://hexhamallstar.github.io/tag/supervised.html>)
 Classification (<https://hexhamallstar.github.io/tag/classification.html>)
 EDA (<https://hexhamallstar.github.io/tag/eda.html>)
 Data Cleaning (<https://hexhamallstar.github.io/tag/data-cleaning.html>)
 Clustering (<https://hexhamallstar.github.io/tag/clustering.html>)
 Unsupervised (<https://hexhamallstar.github.io/tag/unsupervised.html>)
 PCA (<https://hexhamallstar.github.io/tag/pca.html>)
 bokeh (<https://hexhamallstar.github.io/tag/bokeh.html>)
 Collaborative Filtering (<https://hexhamallstar.github.io/tag/collaborative-filtering.html>)
 Kaggle (<https://hexhamallstar.github.io/tag/kaggle.html>)
 Feature Creation (<https://hexhamallstar.github.io/tag/feature-creation.html>)
 Selenium (<https://hexhamallstar.github.io/tag/selenium.html>)
 Webscraping (<https://hexhamallstar.github.io/tag/webscraping.html>)
 Data Visualisation (<https://hexhamallstar.github.io/tag/data-visualisation.html>)
 Derivation (<https://hexhamallstar.github.io/tag/derivation.html>)
 Regression (<https://hexhamallstar.github.io/tag/regression.html>)
 Beginner (<https://hexhamallstar.github.io/tag/beginner.html>)

Site built using Pelican (<http://getpelican.com/>) • Theme based on VoidyBootstrap (<http://www.voidynullness.net/page/voidy-bootstrap-pelican-theme/>) by RKI (<http://www.robertiwancz.com/>)