NBA: Does Player Performance Affect Social Media Presence?

January 15, 2018

```
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')

from client.api.notebook import Notebook
ok = Notebook('exploration.ok')
_ = ok.auth(inline=True)
```

NBA Social Presence and Performance

```
In [3]: from datascience import *

import matplotlib
matplotlib.use('Agg', warn=False)
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
np.set_printoptions(threshold=50)
```

The NBA is an influential organization, and its players are often leaders in social media and following. The NBA creates "celebrities" in a sense, so it is interesting to analyze whether their popularity stems from their performance. The project focuses on using player stats to gain insights about team performance and players' social media engagement.

0.3 1. The Data

Choose and describe a data set that includes at least two tables, some quantitative variable, and some categorical variable.

The dataset, Social Power-NBA, contains data on NBA player stats and social media presence. This data was obtained through Kaggle at https://www.kaggle.com/noahgift/social-power-nba.

The `rpm` data describes the NBA players' and their games played, minutes per game, offensive real plus minus, defensive real plus minus, real plus minus, numbers of wins.

https://deadspin.com/just-what-the-hell-is-real-plus-minus-espns-new-nba-s-1560361469

```
In [4]: import pandas as pd

        rpm = Table.read_table("rpm.csv", sep=",")
        rpm
```

```
Out[4]: NAME                 | TEAM | GP   | MPG  | ORPM | DRPM  | RPM  | WINS
        LeBron James, SF     | CLE  | 74   | 37.8 | 6.49 | 1.93  | 8.42 | 20.43
        Stephen Curry, PG    | GS   | 79   | 33.4 | 7.27 | 0.14  | 7.41 | 18.8
        Jimmy Butler, SG     | CHI  | 76   | 37   | 4.82 | 1.8   | 6.62 | 17.35
        Russell Westbrook, PG | OKC  | 81   | 34.6 | 6.74 | -0.47 | 6.27 | 17.34
        Draymond Green, PF   | GS   | 76   | 32.5 | 1.55 | 5.59  | 7.14 | 16.84
        Rudy Gobert, C       | UTAH | 81   | 33.9 | 0.35 | 6.02  | 6.37 | 15.55
        James Harden, SG     | HOU  | 81   | 36.4 | 6.38 | -1.57 | 4.81 | 15.54
        Kawhi Leonard, SF    | SA   | 74   | 33.4 | 5.83 | 1.25  | 7.08 | 15.53
        Chris Paul, PG       | LAC  | 61   | 31.5 | 5.16 | 2.76  | 7.92 | 13.48
        Nikola Jokic, C      | DEN  | 73   | 27.9 | 4.44 | 2.29  | 6.73 | 13.18
        ... (458 rows omitted)
```

The variables in `social` include more specific variables including field goals, field goal percetage, page views on Wikipedia, number of times the player favorited a tweet, and number of times the player retweeted. Categorical variables include the player's name and the position played by the player.

```
In [5]: social = Table.read_table("socialmedia.csv", sep=",")
        social
```

```
Out[5]: Unnamed: 0 | Rk   | PLAYER           | POSITION | AGE  | MP   | FG   | FGA  | FG%   |
        0          | 1    | Russell Westbrook | PG       | 28   | 34.6 | 10.2 | 24   | 0.425 |
        1          | 2    | James Harden      | PG       | 27   | 36.4 | 8.3  | 18.9 | 0.44  |
        2          | 4    | Anthony Davis     | C        | 23   | 36.1 | 10.3 | 20.3 | 0.505 |
        3          | 6    | DeMarcus Cousins  | C        | 26   | 34.2 | 9    | 19.9 | 0.452 |
        4          | 7    | Damian Lillard    | PG       | 26   | 35.9 | 8.8  | 19.8 | 0.444 |
        5          | 8    | LeBron James      | SF       | 32   | 37.8 | 9.9  | 18.2 | 0.548 |
        6          | 9    | Kawhi Leonard     | SF       | 25   | 33.4 | 8.6  | 17.7 | 0.485 |
```

```
7          | 10   | Stephen Curry     | PG       | 28   | 33.4 | 8.5  | 18.3 | 0.468 |
8          | 11   | Kyrie Irving      | PG       | 24   | 35.1 | 9.3  | 19.7 | 0.473 |
9          | 12   | Kevin Durant      | SF       | 28   | 33.4 | 8.9  | 16.5 | 0.537 |
... (229 rows omitted)
```

In [6]:
```
#nba player statistic table
stat = Table.read_table("statanba.csv").drop(np.arange(2))
stat
```

```
/srv/app/venv/lib/python3.6/site-packages/datascience/tables.py:699: FutureWarning: elementwise
  if i not in exclude and c not in exclude])
```

Out[6]:
```
PLAYER            | POSITION | AGE  | MP   | FG   | FGA  | FG%   | 3P   | 3PA  | 3P%
Russell Westbrook | PG       | 28   | 34.6 | 10.2 | 24   | 0.425 | 2.5  | 7.2  | 0.343
James Harden      | PG       | 27   | 36.4 | 8.3  | 18.9 | 0.44  | 3.2  | 9.3  | 0.347
Isaiah Thomas     | PG       | 27   | 33.8 | 9    | 19.4 | 0.463 | 3.2  | 8.5  | 0.379
Anthony Davis     | C        | 23   | 36.1 | 10.3 | 20.3 | 0.505 | 0.5  | 1.8  | 0.299
DeMar DeRozan     | SG       | 27   | 35.4 | 9.7  | 20.9 | 0.467 | 0.4  | 1.7  | 0.266
DeMarcus Cousins  | C        | 26   | 34.2 | 9    | 19.9 | 0.452 | 1.8  | 5    | 0.361
Damian Lillard    | PG       | 26   | 35.9 | 8.8  | 19.8 | 0.444 | 2.9  | 7.7  | 0.37
LeBron James      | SF       | 32   | 37.8 | 9.9  | 18.2 | 0.548 | 1.7  | 4.6  | 0.363
Kawhi Leonard     | SF       | 25   | 33.4 | 8.6  | 17.7 | 0.485 | 2    | 5.2  | 0.38
Stephen Curry     | PG       | 28   | 33.4 | 8.5  | 18.3 | 0.468 | 4.1  | 10   | 0.411
... (436 rows omitted)
```

In [7]:
```
#nba player statistic table with twitter details
sm = Table.read_table("socialmedianba.csv").drop(np.arange(2)).drop(np.arange(2, 37, 1)
sm
```

```
/srv/app/venv/lib/python3.6/site-packages/datascience/tables.py:699: FutureWarning: elementwise
  if i not in exclude and c not in exclude])
```

Out[7]:
```
PLAYER          | POSITION | PAGEVIEWS | TWITTER_FAVORITE_COUNT | TWITTER_RETWEET_COUNT
Aaron Brooks    | PG       | 10        | 1                      | 3
Aaron Gordon    | SF       | 666       | 42.5                   | 16
Adreian Payne   | PF       | 166       | 0                      | 13
Al Horford      | C        | 870       | 136                    | 71
Al-Farouq Aminu | SF       | 330.5     | 33                     | 9
Alec Burks      | SG       | 111       | 10                     | 15
Alex Len        | C        | 270       | 10.5                   | 6
Allen Crabbe    | SG       | 193.5     | 11.5                   | 9
Alonzo Gee      | SF       | 109       | nan                    | nan
Amir Johnson    | PF       | 282       | 31.5                   | 6
... (228 rows omitted)
```

In [8]:
```
#nba player salary table
#note: bobby portis is listed twice in this table, possibly because he had a salary ra
salary2 = Table.read_table("salarynba.csv").sort("NAME", descending = False)#.drop("PO
salary2
```

```
Out[8]: NAME            | POSITION | TEAM                   | SALARY
        Aaron Brooks    | PG       | Indiana Pacers         | 2700000
        Aaron Gordon    | PF       | Orlando Magic          | 4351320
        Aaron Harrison  | SG       | Charlotte Hornets      | 874636
        Abdel Nader     | SF       | Boston Celtics         | 1167333
        Adreian Payne   | PF       | Minnesota Timberwolves | 2022240
        Al Horford      | C        | Boston Celtics         | 26540100
        Al Jefferson    | C        | Indiana Pacers         | 10314532
        Al-Farouq Aminu | SF       | Portland Trail Blazers | 7680965
        Alec Burks      | SG       | Utah Jazz              | 10154495
        Alex Abrines    | SG       | Oklahoma City Thunder  | 5994764
        ... (439 rows omitted)
```

The above table joins the three tables of data we have for overall game statistics, Twitter social media, and salary by player name. The joined table lets us know that there are 241 players in common among all the tables. We can use the combined data to understand Twitter usage for players that have data recorded for that column. After comparing the previous three tables, the joined tables also allowed us to discover an inconsistency with the data as the joined data total was greater than the number of players in the social media table. By grouping by player name and then using the where function in the original tables, we discovered Bobby Portis is included an additional two times (once in the salary and once in the social media data). This additional data may be due to him receiving a salary increase in the year.

```
In [9]: all_data = stat.join("PLAYER", salary2, "NAME").join("PLAYER" ,sm, "PLAYER").drop("TEAM
        #all_data.where("PLAYER", are.equal_to("Bobby Portis"))
        all_data.show(10)
```

```
<IPython.core.display.HTML object>
```

The above table joins the three tables of data we have for overall game statistics, Twitter social media, and salary by player name. The joined table lets us know that there are 241 players in common among all the tables. We can use the combined data to understand Twitter usage for players that have data recorded for that column. After comparing the previous three tables, the joined tables also allowed us to discover an inconsistency with the data as the joined data total was greater than the number of players in the social media table. By grouping by player name and then using the where function in the original tables, we discovered Bobby Portis is included an additional two times (once in the salary and once in the social media data). This additional data may be due to him receiving a salary increase in the year.

```
In [10]: bobbies = all_data.where("PLAYER", are.equal_to("Bobby Portis"))
         bobbies
```

```
Out[10]: PLAYER       | POSITION | AGE  | MP   | FG   | FGA  | FG%   | 3P   | 3PA  | 3P%   | 2P
         Bobby Portis | PF       | 21   | 15.6 | 2.9  | 5.9  | 0.488 | 0.5  | 1.5  | 0.333 | 2
         Bobby Portis | PF       | 21   | 15.6 | 2.9  | 5.9  | 0.488 | 0.5  | 1.5  | 0.333 | 2
```

Below we have removed Bobby from the data as we do not know which social media data to pair up with his salary data. Since this information will be used in our prediction testing later, we will also leave him out so that it will keep our data more consistent.

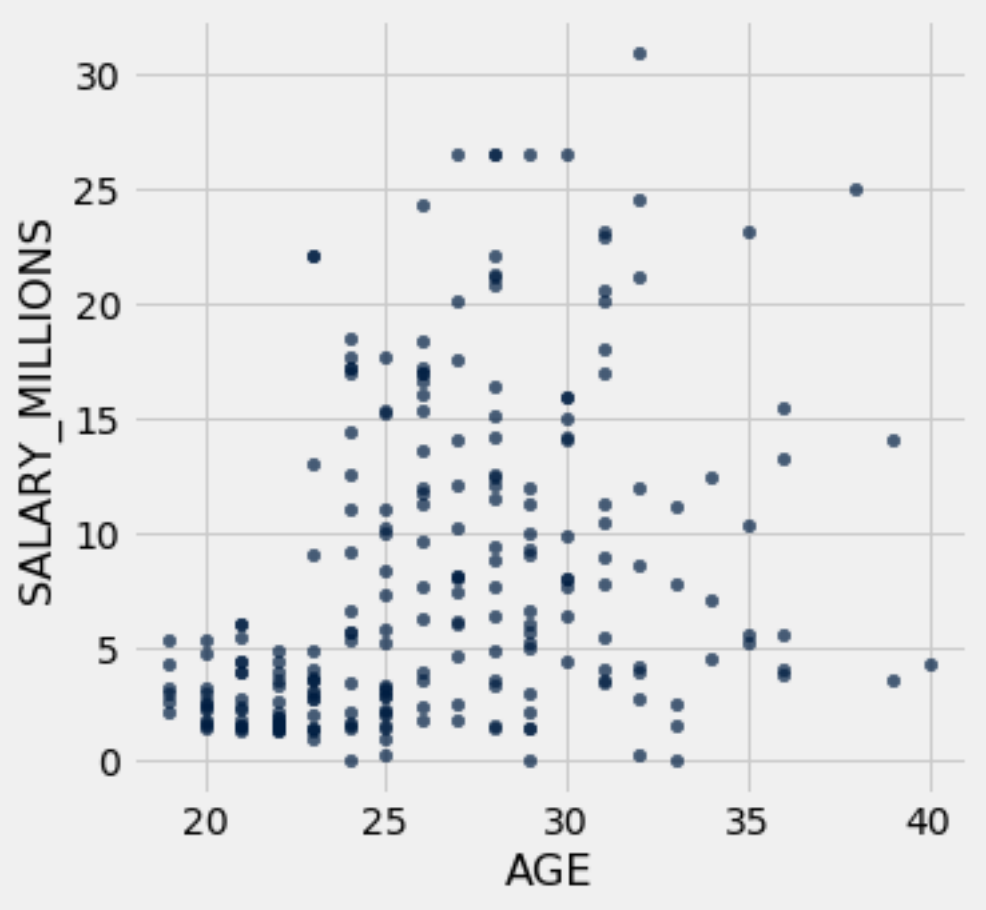
```
In [11]: cleaned_all_data = all_data.where("PLAYER", are.not_equal_to("Bobby Portis"))
         cleaned_all_data.show(5)
```

```
<IPython.core.display.HTML object>
```

0.4 2. Quantitative Visualization

It is interesting to develop visualizations about player attributes and compensation. Below is a scatterplot of a player's annual salary against the player's age.

```
In [12]: social.scatter('AGE', 'SALARY_MILLIONS')
```



We can see that there does not seem to be great correlation between the two vairbales, as the data points form a cloud. The exception is for young players, who do not see such variability in their salaries. As a player becomes older, we might infer that salary is determined better by factors other than age. That is, could performance be a better determinant of a player's salary?

A Below, we plotted RPM (real plus-minus) as a regressor and the salary on the vertical axis.

What is a good determinant of player performance?

We considered several vairables that may be reasonable metrics for player performance. Some of these included average points per game, assists, free throw percentage and field goal percentage. The problem with these variables is that they do not account for a player's position and the time they have on the court. For example, because a point guard normally leads their team in assists and steals, they will likely have a larger number of assists than players of other positions. Let's see if this is true.

```
In [13]: social.group('POSITION', np.mean).select('POSITION', 'FG mean', 'STL mean', 'AST mean
```

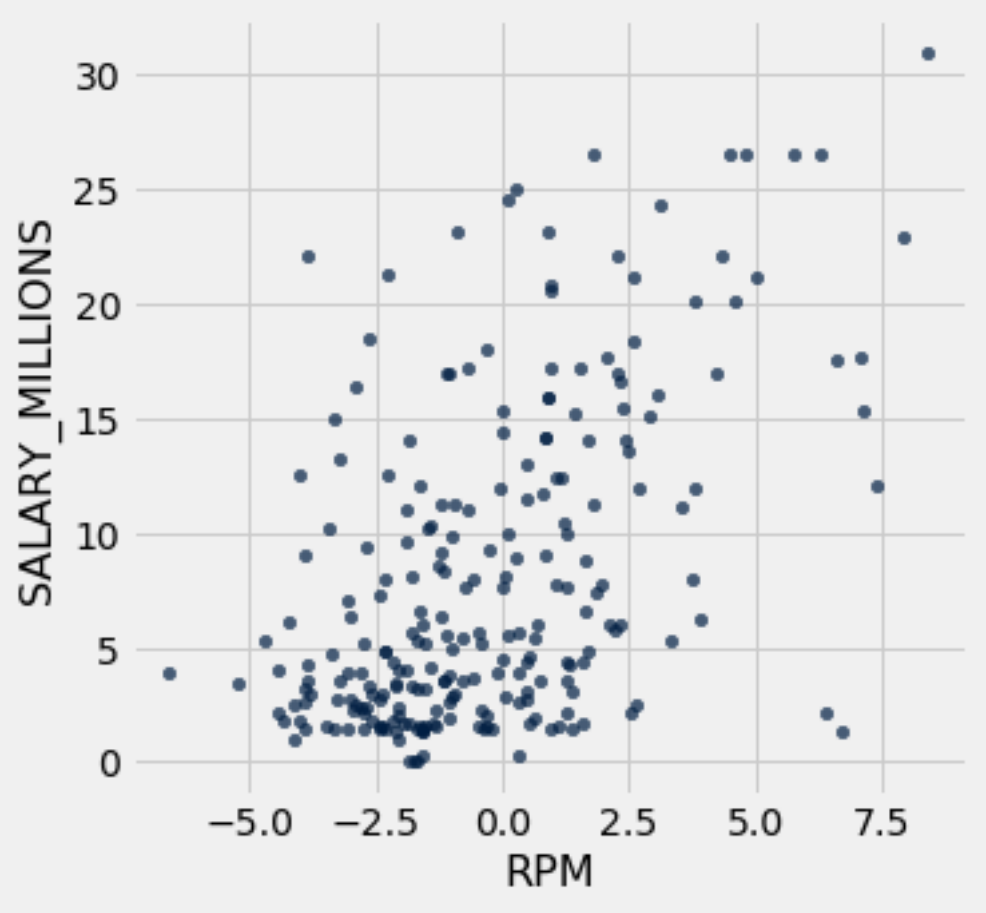
```
Out[13]: POSITION | FG mean | STL mean | AST mean
         C        | 4.01163 | 0.637209 | 1.54884
         PF       | 3.225   | 0.557692 | 1.33462
         PG       | 4.258   | 0.928    | 4.462
         SF       | 3.9587  | 0.9      | 2.06304
         SG       | 3.82083 | 0.785417 | 2.07292
```

As the table shows, point guards have a much larger assist mean than other players and also have larger average STL than other players. Therefore, these metrics are not suitable to compare a player's performance on the field.

Instead a metric, called RPM (real plus-minus), "isolates the unique plus-minus impact of each NBA player by adjusting for the effects of each teammate, opposing player and coach, etc. The RPM model sifts through more than 230,000 possessions each NBA season to tease apart the "real" plus-minus effects attributable to each player, employing techniques similar to those used by scientific researchers when they need to model the effects of numerous variables at the same time. It reflects the impact of each player on his team's scoring margin after controlling for the strength of every teammate and every opponent during each minute he's on the court."

We will use RPM as a better indicator of how well a player is playing when he is on the court. Let us see if there is an association between RPM and players' salaries.

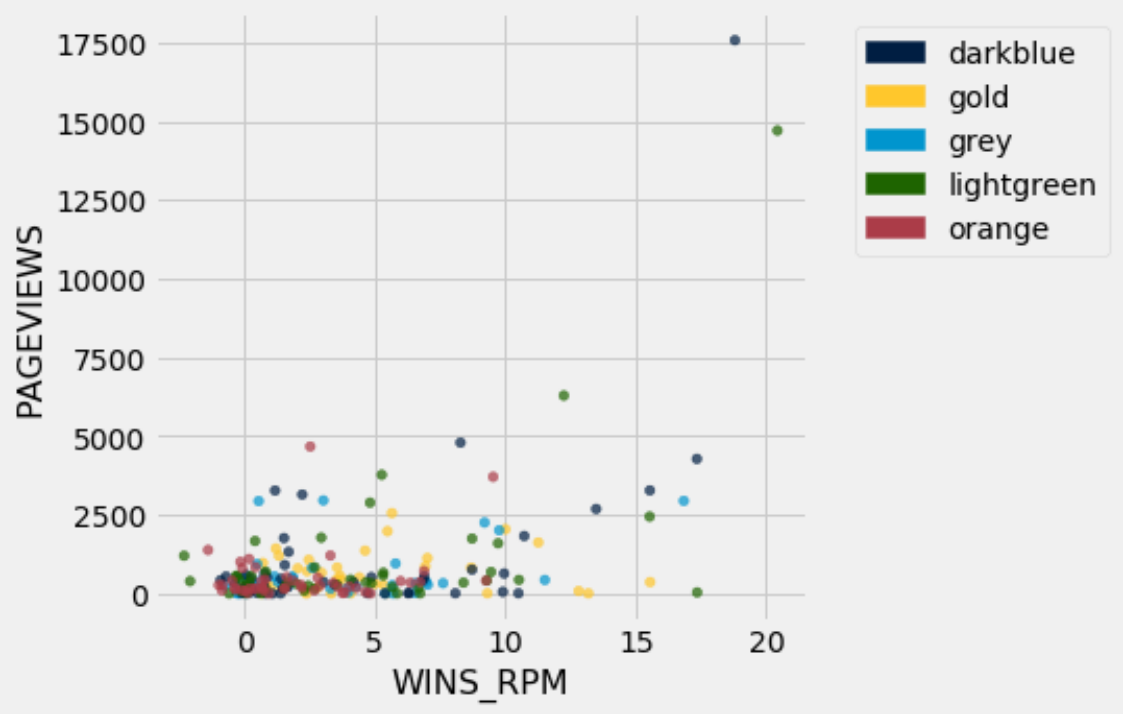
```
In [14]: social.scatter('RPM','SALARY_MILLIONS')
```




```
In [15]: social.group('POSITION')
```

```
Out[15]: POSITION | count
         C        | 43
         PF       | 52
         PG       | 50
         SF       | 46
         SG       | 48
```

```
In [26]: color_table = Table().with_columns(
             'POSITION', make_array('PG', 'C', 'PF', 'SF', 'SG'),
             'Color', make_array('darkblue', 'gold', 'grey', 'lightgreen', 'orange')
         )
         social = social.join('POSITION', color_table)
         social
         win_views = social.scatter('WINS_RPM', 'PAGEVIEWS', colors='Color')
```



As we see, there are two players who have a much larger number of daily page views than other NBA players. Who are they?

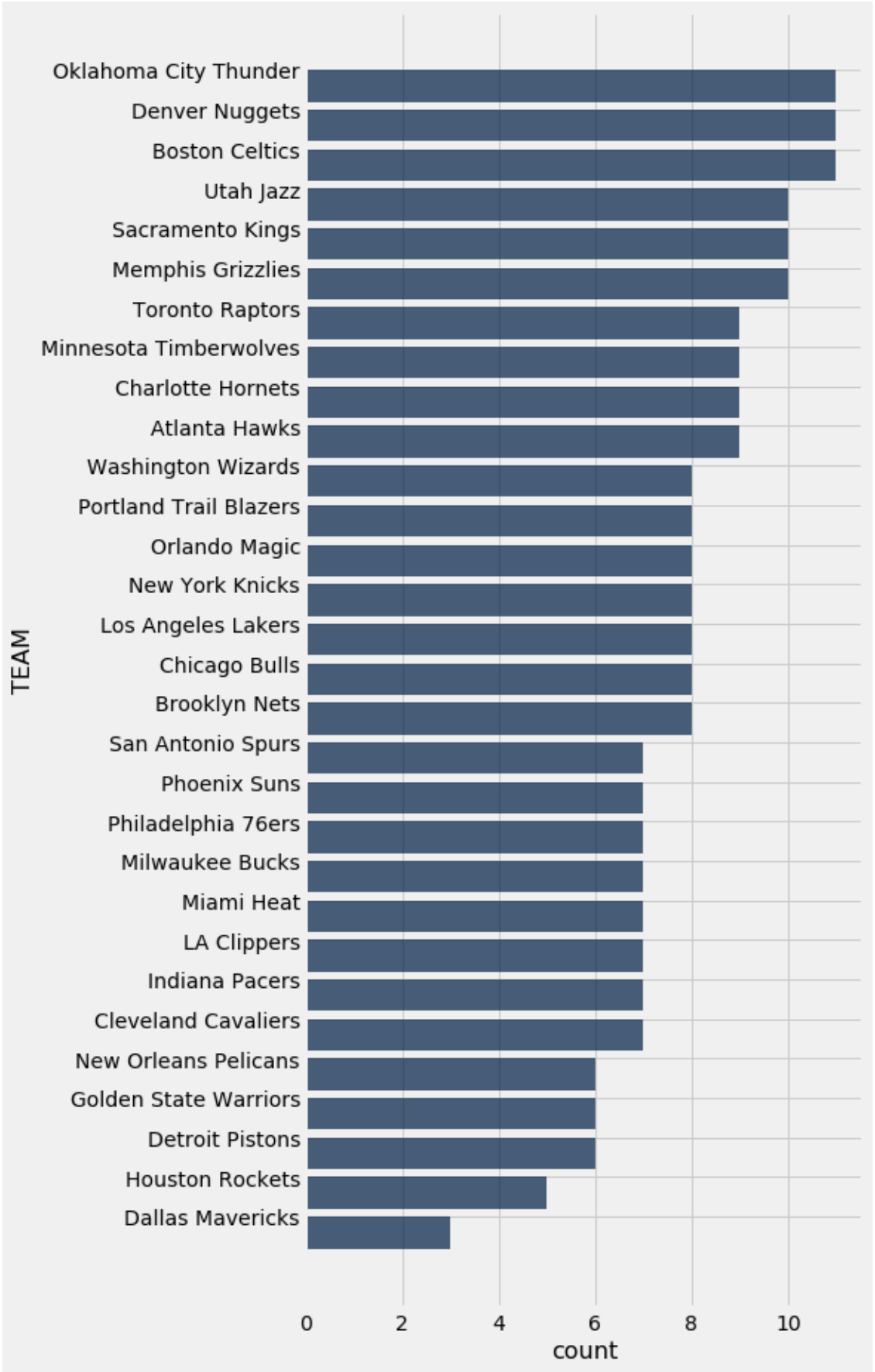
```
In [17]: social.where('PAGEVIEWS', are.above(12500)).select('PLAYER','PAGEVIEWS','WINS_RPM')
```

```
Out[17]: PLAYER        | PAGEVIEWS | WINS_RPM
         Stephen Curry | 17570.5   | 18.8
         LeBron James  | 14704     | 20.43
```

0.5 3. Qualitative Visualization

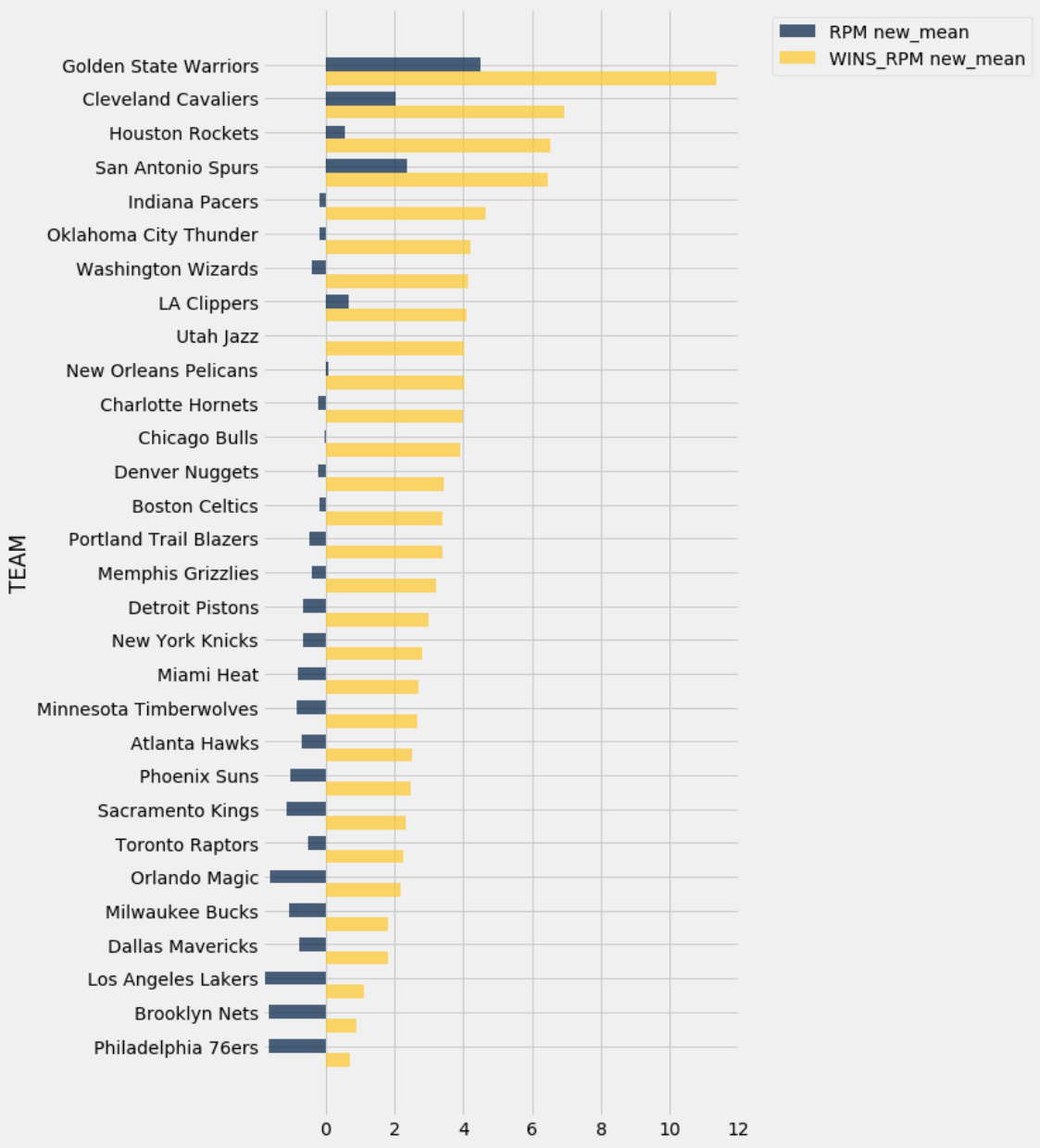
Below we have categorized the cleaned data by team. This allows us to see where most of our players are coming from that also use Twitter. On average it looks like we have about six players per team, with the most being from Oklahoma City Thunder, Denver Nuggets, and Boston Celtics, and the fewest from Dallas Mavericks.

```
In [18]: team_category = cleaned_all_data.group("TEAM").sort("count", descending = True)
         team_category.barh("TEAM")
```


Oklahoma City Thunder
Denver Nuggets
Boston Celtics
Utah Jazz
Sacramento Kings
Memphis Grizzlies
Toronto Raptors
Minnesota Timberwolves
Charlotte Hornets
Atlanta Hawks
Washington Wizards
Portland Trail Blazers
Orlando Magic
New York Knicks
Los Angeles Lakers
Chicago Bulls
Brooklyn Nets
San Antonio Spurs
Phoenix Suns
Philadelphia 76ers
Milwaukee Bucks
Miami Heat
LA Clippers
Indiana Pacers
Cleveland Cavaliers
New Orleans Pelicans
Golden State Warriors
Detroit Pistons
Houston Rockets
Dallas Mavericks
TEAM
0
2
4
6
8
10
count

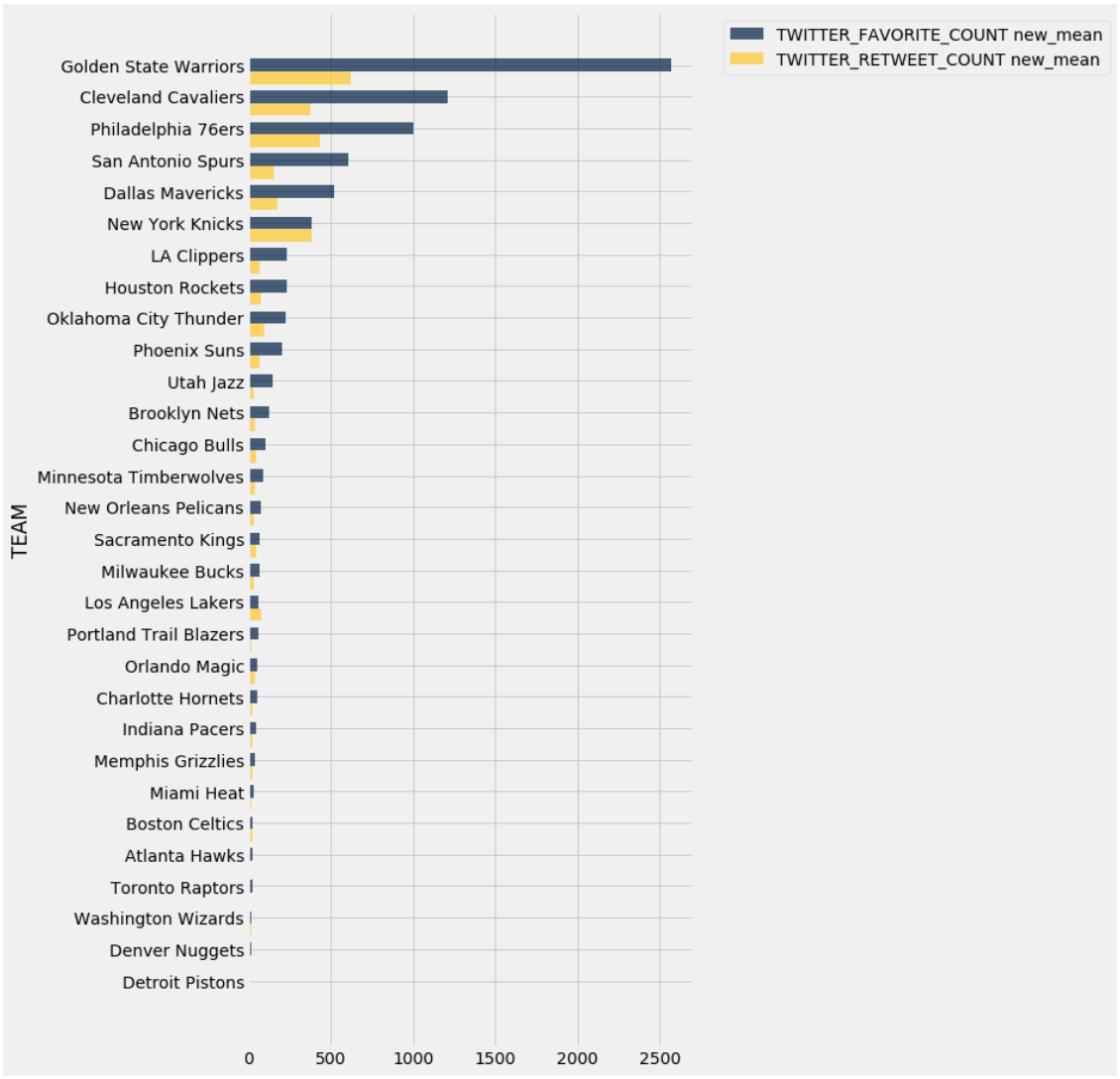
Below we have an overlapped graph of player teams and RPM and WINS_RPM means. This allows us to see how successful each team is and which teams are more likely to have successful players. As we can see, teams with more wins, have higher RPMs while teams who have a low WIN_RPM is likely to have a negative RPM value for their players.

```
In [19]: import math
         def new_mean(array):
             "Edited np.mean function to account for nans in array"
             new_array = make_array()
             value = 0

             length = len(array)
             new_length = len(array)
             for i in np.arange(length):
                 if(math.isnan(array.item(i))):
                     new_length = new_length - 1
                 else:
                     value = array.item(i)
                     new_array = np.append(new_array, value)
             mean = np.average(new_array)
             return mean
         team_game_stats = cleaned_all_data.group("TEAM", new_mean).drop("PLAYER new_mean", "P(
         team_game_stats.barh("TEAM")
```



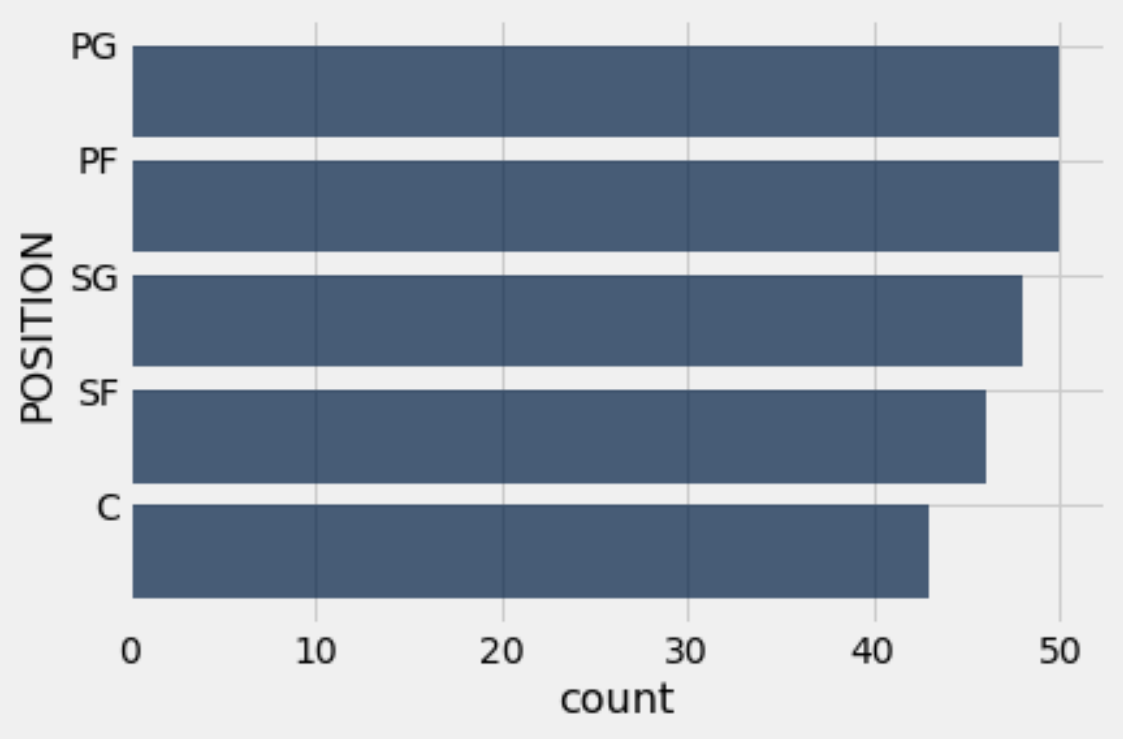
This is a categorical graph showing the mean twitter favorite and and retweet mean among teams. Similar to the RPM team graph, we see a faint connection in the order of teams. Interestingly, the Philadelphia 76ers, which had the lowest WINS_RPM mean, has the third highest Twitter Favorite Count mean.

```
In [20]: team_sm_stats = cleaned_all_data.group("TEAM", new_mean).drop("PLAYER new_mean", "POS
         team_sm_stats.barh("TEAM")
```



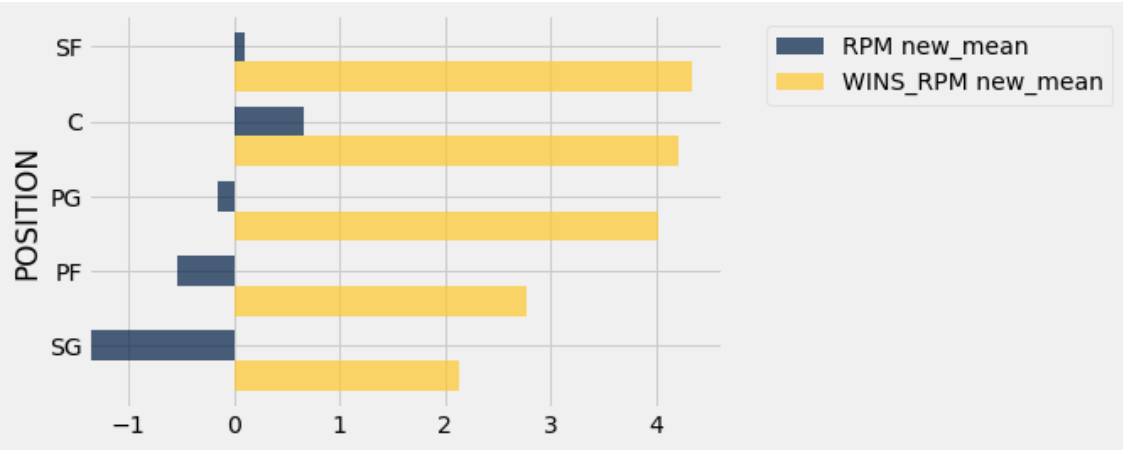
We also have a categorical graph displaying the number of different positions the players in the nba have. There appears to be a fairly even distribution of positions.

```
In [21]: position_category = cleaned_all_data.group("POSITION").sort("count", descending = True
         position_category.barh("POSITION")
```



Measuring RPM means and RPM_win means with position, we can see there may not be a strong connection between positions and RPM because the bin is so small (within a range of about 5).

```
In [22]: position_game_stats = cleaned_all_data.group("POSITION", new_mean).drop("PLAYER new_m
         position_game_stats.barh("POSITION")
```



Interestingly, we only have Twitter data for point guards and shooting guards. Tehe means of favorites and rewtweets for point guards far exceeds the amount that shooting guards have.

```
In [ ]: position_sm_stats = cleaned_all_data.group("POSITION", new_mean).drop("PLAYER new_mean
        position_sm_stats.barh("POSITION")
```

0.6 4. Grouping/ Pivot Tables

Let's see how teams rank by performance. If we use RPM as a metric for an individuals performance, a team will a low average RPM will likely do worse than a team with a high average RPM. This can be argued, because one very talented player can nudge the average RPM up and cause average RPM to misrepresent the performance of the team. However, though basketball is a team sport, individual's contributions are still important.

In order to group average RPM by team, we want to have the full team names (for clarity and those who do not know ach teams abbreviation). Further, the current team abbbreviations for some players include two teams based on if they have played for multiple teams. We only want the team they are currently playingon (2016-2017 season) Let's join a few tables:

```
In [ ]: salary = Table.read_table("salarynba.csv", sep=",")
        salary
```

Below we have adjusted RPM by multiplying by 1000, so that it is visible in the visualization next to average page views per team.

```
In [ ]: full_teams = social.join('PLAYER',salary, 'NAME')
        array_adj_RPM = full_teams.column('RPM')*1000
        full_teams2 = full_teams.with_column('Adjusted RPM',array_adj_RPM)
        full_teams2.group('TEAM_2', np.mean).select('TEAM_2','Adjusted RPM mean','PAGEVIEWS mea
```

It looks like teams that perform better are more popular in terms of page views on Wikipedia! We didn't expect this for the Lakers, since the Lakers brand is quite popular despite the team's low performance.

0.7 Player Performance and RPM: Is there an association?

The previous visualizations have interested us in understanding whether player performance has an effect on social media presence.

Our question: Are players that are objectively "better" at basketball more likely to engage on social media, particularly Twitter?

1. **Sample:** 2016-2017 NBA basketball players
2. **Population:** All professional basketball players
3. **Metrics:** Social Media Engagement (sum of # of favorites and # of retweets made by player in given year), Player Performance (RPM)

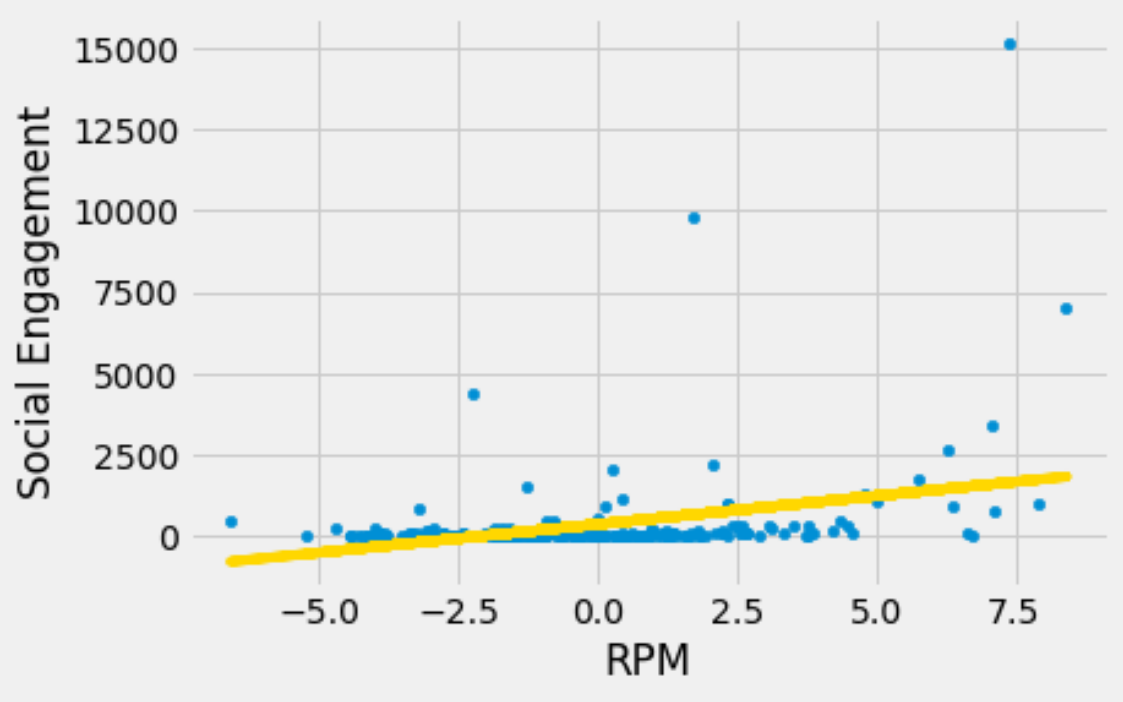
0.8 6. Hypothesis Testing

Null Hypothesis: There is no correlation between player's performance on social media engagement. (slope of the true line =0)

Alternative Hypothesis: There is some correlation between a player's performance on social media engagement. (slope of the true line does not equal 0)

```
In [30]: engagement = Table().with_columns('Player', social.column('PLAYER'),
                                           'RPM',social.column('RPM'),
                                           'Social Engagement', social.column('TWITTER_FAVORITE
         df = {'RPM': social.column('RPM'), 'Social Engagement': social.column('TWITTER_FAVORI
         engagementdf = pd.DataFrame(data=df)
         engagement = engagementdf.fillna(0)
         engagement = Table.from_df(engagement)
         engagement

Out[30]: RPM   | Social Engagement
         4.35  | 472
         4.2   | 193.5
         2.13  | 105
         1.7   | 9793.5
         2.58  | 75.5
         6.73  | 1
         0.77  | 41
         2.66  | 84
         -0.68 | 342
         6.37  | 956.5
         ... (229 rows omitted)

In [31]: def standard_units(any_numbers):
             "Convert any array of numbers to standard units."
             return (any_numbers - np.mean(any_numbers))/np.std(any_numbers)

         def correlation(t, x, y):
             return np.mean(standard_units(t.column(x))*standard_units(t.column(y)))

         def slope(table, x, y):
             r = correlation(table, x, y)
             return r * np.std(table.column(y))/np.std(table.column(x))

         def intercept(table, x, y):
             a = slope(table, x, y)
             return np.mean(table.column(y)) - a * np.mean(table.column(x))

         def fit(table, x, y):
             a = slope(table, x, y)
             b = intercept(table, x, y)
             return a * table.column(x) + b

         def residual(table, x, y):
             return table.column(y) - fit(table, x, y)

         def scatter_fit(table, x, y):
             plots.scatter(table.column(x), table.column(y), s=20)
             plots.plot(table.column(x), fit(table, x, y), color='gold')
```

```
    plots.xlabel(x)
    plots.ylabel(y)

scatter_fit(engagement, 'RPM', 'Social Engagement')
```




```
In [32]: correlation(engagement, 'RPM', 'Social Engagement')
```

```
Out[32]: 0.34806737944897265
```

```
In [33]: slope(engagement, 'RPM', 'Social Engagement')
```

```
Out[33]: 173.68813385028059
```

0.9 7. Statistical Test

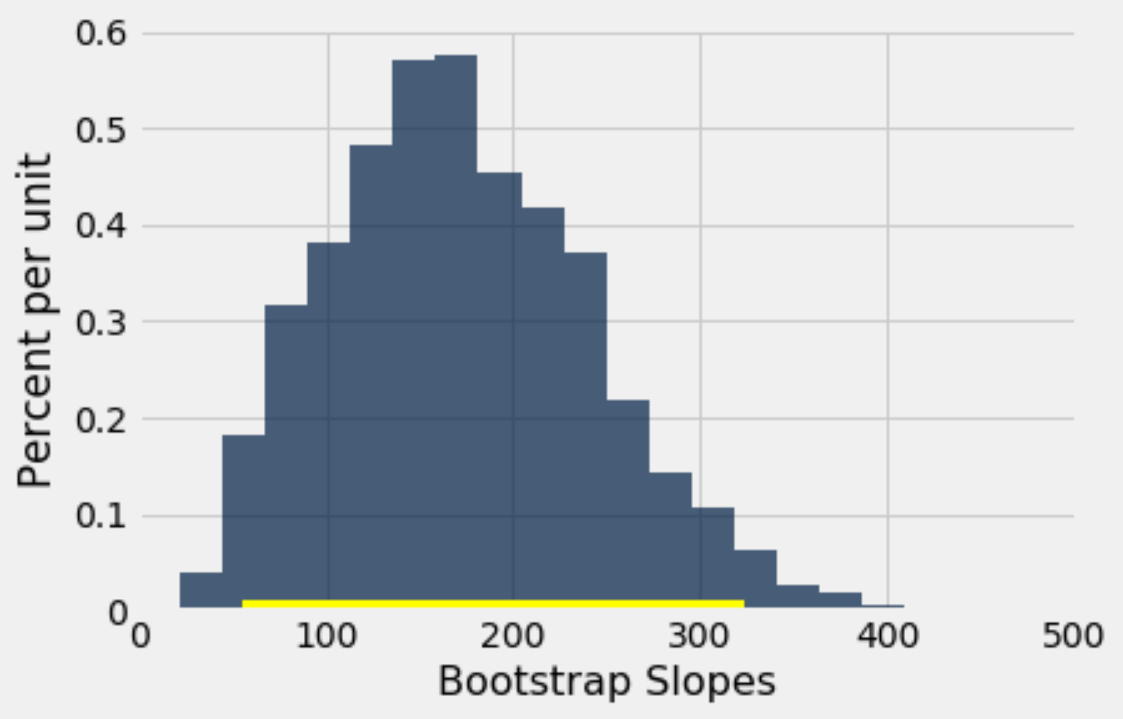
Bootstrapping (Estimating the True Slope)

```
In [34]: def bootstrap_slope(table, x, y, repetitions):

             # For each repetition:
             # Bootstrap the scatter, get the slope of the regression line,
             # augment the list of generated slopes
             slopes = make_array()
             for i in np.arange(repetitions):
                 bootstrap_sample = table.sample()
                 bootstrap_slope = slope(bootstrap_sample, x, y)
```

```
        slopes = np.append(slopes, bootstrap_slope)

    # Find the endpoints of the 95% confidence interval for the true slope
    left = percentile(2.5, slopes)
    right = percentile(97.5, slopes)

    # Slope of the regression line from the original sample
    observed_slope = slope(table, x, y)

    # Display results
    Table().with_column('Bootstrap Slopes', slopes).hist(bins=20)
    plots.plot(make_array(left, right), make_array(0, 0), color='yellow', lw=8);
    print('Slope of regression line:', observed_slope)
    print('Approximate 95%-confidence interval for the true slope:')
    print(left, right)
```

```
In [35]: bootstrap_slope(engagement, 'RPM', 'Social Engagement', 5000)
```

```
Slope of regression line: 173.68813385
Approximate 95%-confidence interval for the true slope:
55.5684461892 323.290277758
```



Because the confidence interval does not contain 0, we reject the null hypothesis that the true slope is 0 and conclude that there is an association between RPM and Social Media Engagement at 5% significance.

0.10 8. Prediction: Can we predict a player's position based on his stats?

In [36]:
```
#split data into train and test
cleaned_social = social.drop(1,2,'PLAYER','AGE','TEAM','Color','PACE','PAGEVIEWS','TW
shuffled_social = cleaned_social.sample(with_replacement=False)
training_set = shuffled_social.take(np.arange(89))
test_set  = shuffled_social.take(np.arange(89, 178))
test_set
```

Out[36]:

POSITION	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	el
PG	22	4.1	9.8	0.414	1.5	4.2	0.358	2.5	5.6	0.456	0
PF	15.6	2.9	5.9	0.488	0.5	1.5	0.333	2.4	4.4	0.541	0
PF	14.8	2.5	5.1	0.484	0.5	1.5	0.309	2	3.6	0.556	0
C	8.6	0.7	1.4	0.523	0	0	nan	0.7	1.4	0.523	0
SG	18.7	2.5	6.4	0.39	1.3	3.3	0.392	1.2	3.1	0.387	0
SF	33.4	8.6	17.7	0.485	2	5.2	0.38	6.6	12.5	0.529	0
SG	27.4	3.1	7.9	0.396	1.7	4.5	0.368	1.5	3.4	0.433	0
SF	21.7	3	6.7	0.45	1	3.2	0.321	2	3.4	0.571	0
SG	14.6	2	5.1	0.389	0.8	2.7	0.308	1.2	2.4	0.479	0
SF	32.4	4.6	10	0.463	2.2	5.5	0.398	2.4	4.5	0.54	0

... (79 rows omitted)

In [37]:
```
#implementing classifier
def distance(point1, point2):
    """Returns the distance between point1 and point2
    where each argument is an array
    consisting of the coordinates of the point"""
    return np.sqrt(np.sum((point1 - point2)**2))

def all_distances(training, new_point):
    """Returns an array of distances
    between each point in the training set
    and the new point (which is a row of attributes)"""
    attributes = training.drop('POSITION')
    def distance_from_point(row):
        return distance(np.array(new_point), np.array(row))
    return attributes.apply(distance_from_point)

def table_with_distances(training, new_point):
    """Augments the training table
    with a column of distances from new_point"""
    return training.with_column('Distance', all_distances(training, new_point))

def closest(training, new_point, k):
    """Returns a table of the k rows of the augmented table
    corresponding to the k smallest distances"""
    with_dists = table_with_distances(training, new_point)
    sorted_by_distance = with_dists.sort('Distance')
```

```
            topk = sorted_by_distance.take(np.arange(k))
            return topk
```

```
In [38]: #gets the majority
         def classify(training, new_point, k):
             closestk = closest(training, new_point, k)
             max_position = closestk.group('POSITION').sort('count',descending = True).column(
             return max_position
```

```
In [39]: classify(test_set, cleaned_social.drop('POSITION').row(31), 10)
```

```
Out[39]: 'PF'
```

```
In [40]: #proportion correct
         def count_zero(array):
             """Counts the number of 0's in an array"""
             return len(array) - np.count_nonzero(array)

         def count_equal(array1, array2):
             """Takes two numerical arrays of equal length
             and counts the indices where the two are equal"""
             return array1 == array2

         def evaluate_accuracy(training, test, k):
             test_attributes = test_set.drop('POSITION')
             def classify_testrow(row):
                 return classify(training, row, k)
             c = test_attributes.apply(classify_testrow)
             print(c)
             return np.count_nonzero(count_equal(c, test.column('POSITION')))/test.num_rows
```

```
In [41]: evaluate_accuracy(training_set, test_set, 8)
```

```
['PG' 'SG' 'SG' ..., 'PF' 'SG' 'SF']


Out[41]: 0.29213483146067415
```

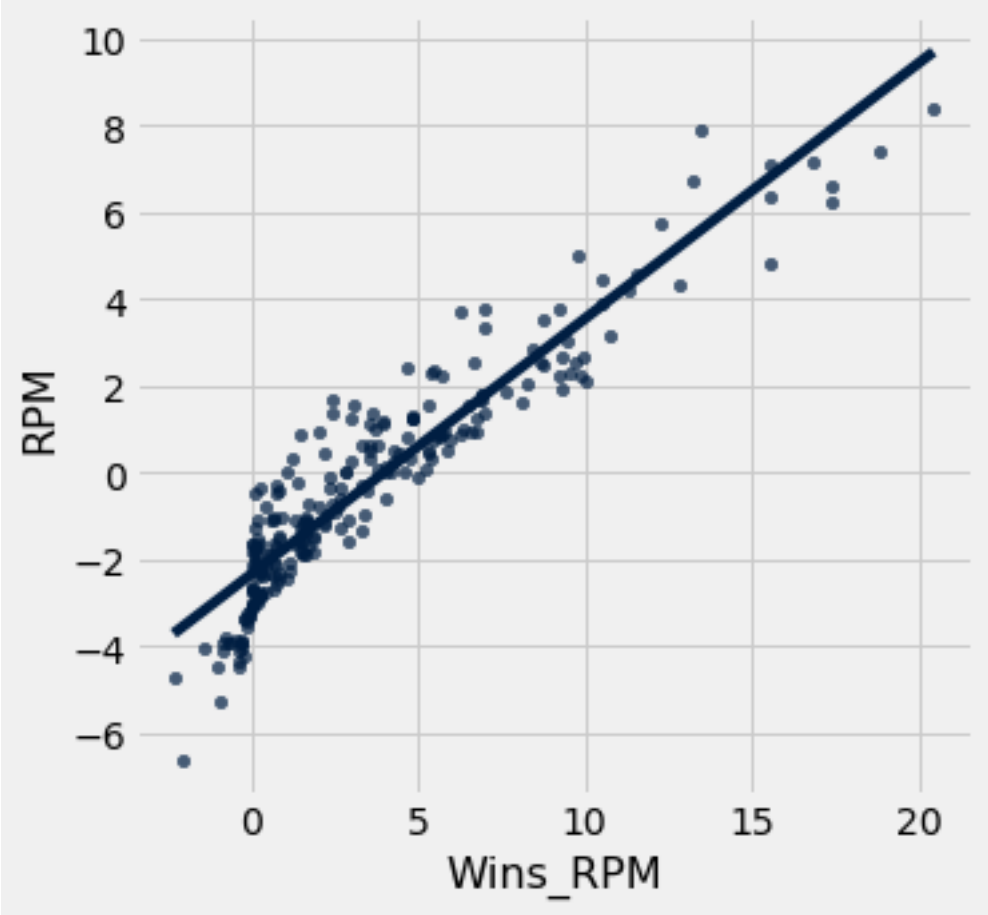
0.11 Another Prediction Problem

Since there appears to be a some sort of relationship between RPM and Wins_RPM based on the categorical overlaid graph, we would like to be able to predict what a player's Wins_RPM will be based off of their RPM. Players with high RPM are likely to contribute more to their team's Wins_RPM, so it being able to predict RPM will be a good way to measure a player's value to their team.

Since there is an association between this quantitative data, we will use a best fit regression line. The best fit regression line is a good choice because once we have create a function based off of the fit line, we can always estimate what the RPM will be using Wins_RPM.

In [42]:
```
#functions needed for prediction

def standard_units(array):
    return (array - np.mean(array)) /np.std(array)

def correlation(table):
    return np.mean(standard_units(table.column(0)) * standard_units(table.column(1)))

def fit_line(table):
    x = table.column(1)
    y = table.column(0)
    r = correlation(table)
    slope = r * (np.std(y) / np.std(x))
    intercept = np.average(y) - slope * np.average(x)
    return make_array(slope, intercept)

#function needed to fix nan in table
def no_nan(table):
    #"Function to account for remove nans in a table using a two column table,
    #"the nan column must be second column"
    new_RPM_array = make_array()
    new_twitter_array = make_array()
    value = 0
    value_2 = 0

    length = table.num_rows

    for i in np.arange(length):
        if(math.isnan(table.column(1).item(i))):
            test = 123456 #this line is unneccesary
        else:
            value = table.column(0).item(i)
            value_2 = table.column(1).item(i)
            new_RPM_array = np.append(new_RPM_array, value)
            new_twitter_array = np.append(new_twitter_array, value_2)

    new_table = Table().with_columns("RPM", new_RPM_array,
                                     "x value", new_twitter_array)
    return new_table
```

In [43]:
```
#table of x values (RPM) and y values (wins rpm)
data_table = cleaned_all_data.select("RPM", "WINS_RPM")

prediction_with_winrpm = no_nan(data_table).relabel("x value", "Wins_RPM")
prediction_with_winrpm.scatter(1, fit_line = True)
```




```
In [44]: fit_line(prediction_with_winrpm)
```

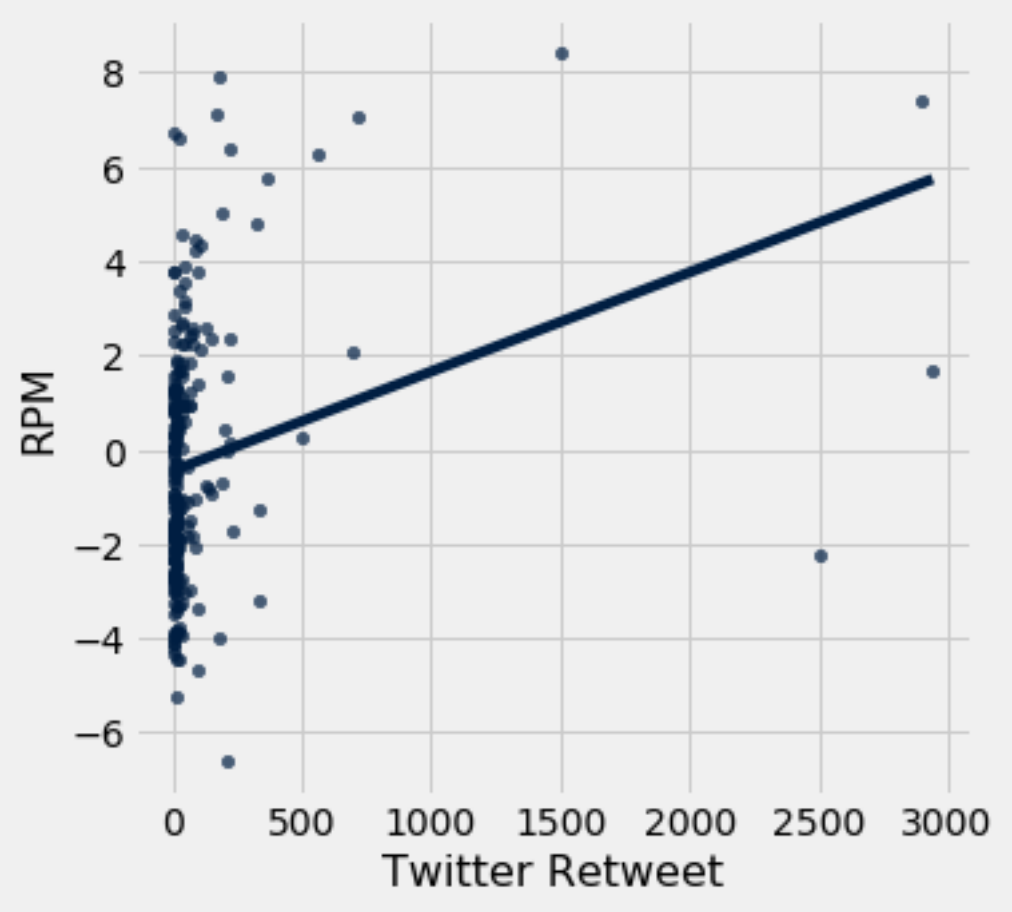
```
Out[44]: array([ 0.58931324, -2.33407397])
```

```
In [45]: #fit value estimator
         def fitted_value(table, given_x):
             slope_and_intercept = fit_line(table)
             new_value = slope_and_intercept.item(0) * given_x + slope_and_intercept.item(1)
             return new_value

         print("RPM Prediction of Win_RPM of 5:", fitted_value(prediction_with_winrpm, 5))
         print("RPM Prediction of Win_RPM of 10:", fitted_value(prediction_with_winrpm, 10))
         print("RPM Prediction of Win_RPM of 15:", fitted_value(prediction_with_winrpm, 15))
         print("RPM Prediction of Win_RPM of 20:", fitted_value(prediction_with_winrpm, 20))
```

```
RPM Prediction of Win_RPM of 5: 0.6124922383854101
RPM Prediction of Win_RPM of 10: 3.5590584469725637
RPM Prediction of Win_RPM of 15: 6.505624655559718
RPM Prediction of Win_RPM of 20: 9.452190864146871
```

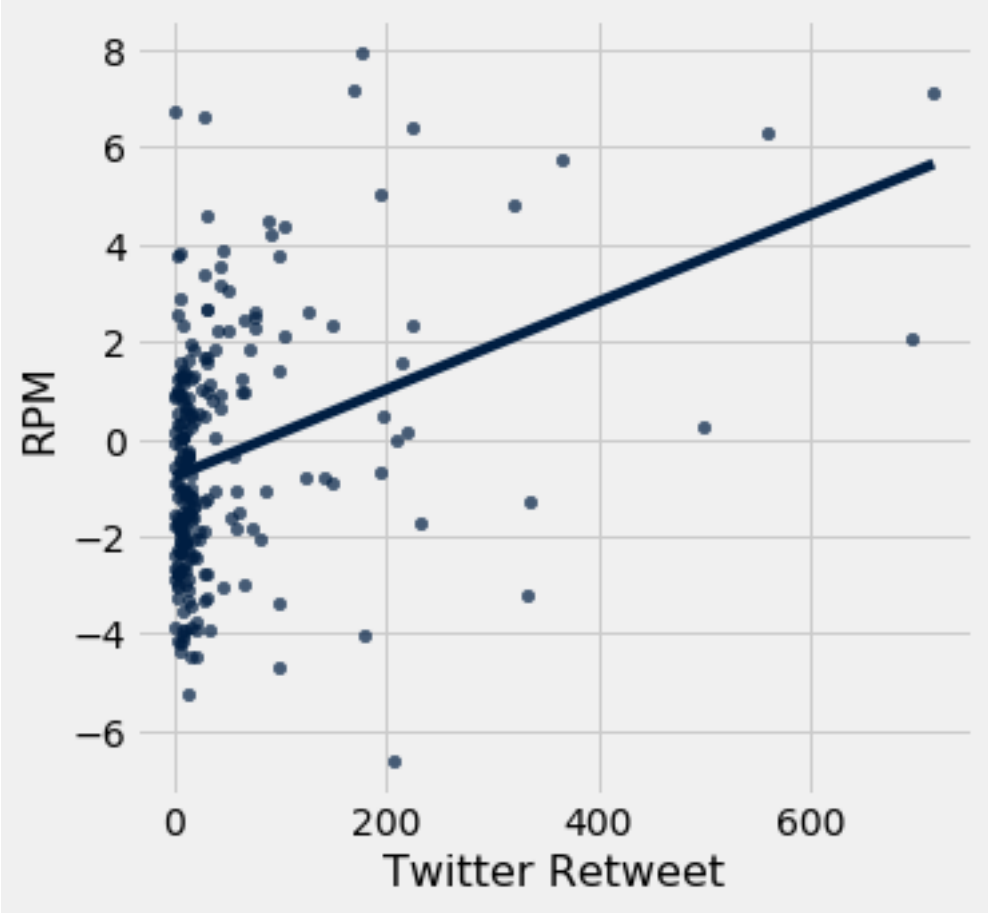
Our fitted value function can now predict y values (RPM) using Wins_RPM. We will also test RPM using Twitter data since we have found an association between the two.

```
In [46]: #table of x values (RPM) and y values (twitter values)
         data_table = cleaned_all_data.select("RPM", "TWITTER_RETWEET_COUNT")

         prediction_with_retweets = no_nan(data_table).relabel("x value", "Twitter Retweet")
         prediction_with_retweets.scatter(1, fit_line = True)
```



Since the there seem to be some outliers, we create a better graph and fit line to capture more of the average by removing the points with a retweet value above 800.

```
In [47]: better_prediction_with_retweets = no_nan(data_table).relabel("x value", "Twitter Retwe
         better_prediction_with_retweets.scatter(1, fit_line = True)
```



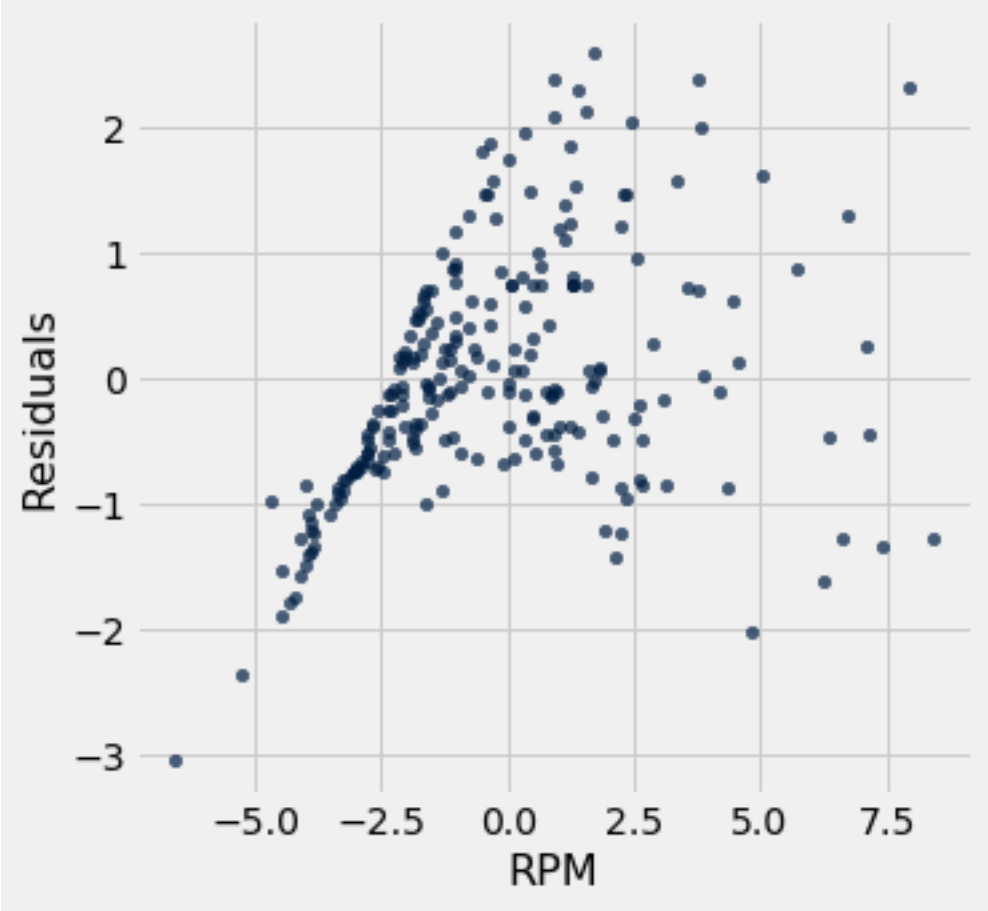

```
In [48]: fit_line(better_prediction_with_retweets)

Out[48]: array([ 0.00897228, -0.7881878 ])

In [49]: print("RPM Prediction of Retweet of 10:", fitted_value(better_prediction_with_retweets
         print("RPM Prediction of Retweet of 50:", fitted_value(better_prediction_with_retweets
         print("RPM Prediction of Retweet of 100:", fitted_value(better_prediction_with_retwee
         print("RPM Prediction of Retweet of 500:", fitted_value(better_prediction_with_retwee

RPM Prediction of Retweet of 10: -0.6984649630237058
RPM Prediction of Retweet of 50: -0.33957359993960773
RPM Prediction of Retweet of 100: 0.10904060391551484
RPM Prediction of Retweet of 500: 3.6979542347564953
```

We will use a residual graph to determine the effectiveness of our regression line

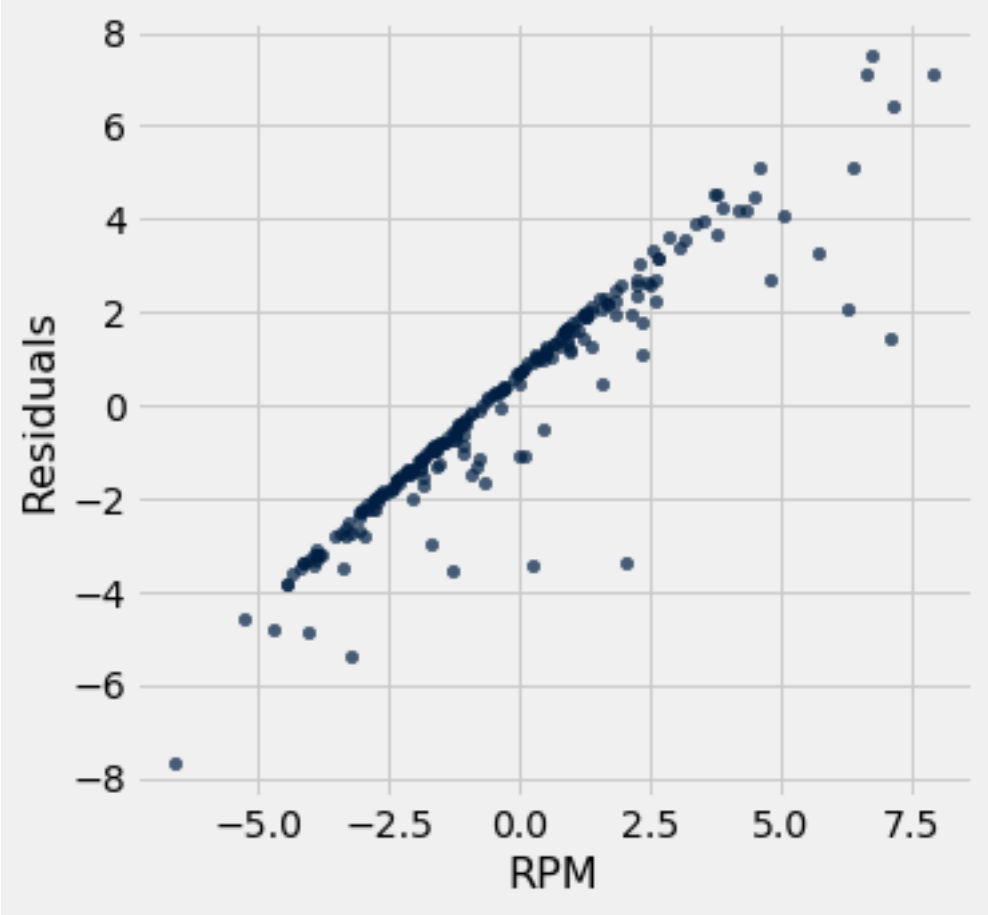
```
In [50]: #residual function
         def residuals(table):
```

```
    slope_intercept = fit_line(table)
    regression_estimates = slope_intercept.item(0) * table.column(1) + slope_intercept
    residuals = table.column(0) - regression_estimates
    return residuals
```

In [51]:
```
prediction_with_winrpm.with_column("Residuals", residuals(prediction_with_winrpm)).dr
```



In [55]:
```
better_prediction_with_retweets.with_column("Residuals", residuals(better_prediction_
```




```
In [56]: residual_sd = np.std(residuals(better_prediction_with_retweets))
         residual_sd
```

```
Out[56]: 2.400386008760155
```

The residual graph for retweets is not random. The residual graph leans in a positive slope and appears biased in its values. When looking at the residual SD, the closer the residuals are to zero, the less the error in regression becomes. Since 2.4 is quite far from zero, visually and calculatively we can tell that estimating RPM from retweets is not that reliable.

0.12 2. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission.

```
In [ ]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>
```