Machine learning model to

Predicate Social-Media Influence in sports

Sports is a fascinating topic for data scientists because there is always a story behind every number. Just because an NBA player scores more points than another player, it doesn’t necessarily mean he adds more value to the team. As a result, there has been a recent explosion in individual statistics that try to measure a player’s impact. ESPN created the Real Plus-Minus, FiveThirtyEight came up with the CARMELO NBA Player Projections, and the NBA has the Player Impact Estimate. Social media is no different; there is more to the story than just a high follower count.

This chapter will explore the numbers behind the numbers using ML and then creating an API to serve out the ML model. All of this will be done in the spirit of solving real-world problems in a real- world way. This means covering details like setting up your environment, deployment, and monitoring, in addition to creating models on clean data.

# Phrasing the Problem

Coming from a cold start in looking at social media and the NBA, there many interesting questions to ask. Here are some examples.

Does individual player performance impact a team’s wins?

Does on-the-court performance correlate with social-media influence? Does engagement on social media correlate with popularity on Wikipedia?

Is follower count or social-media engagement a better predictor of popularity on Twitter? Does salary correlate with on-the-field performance?

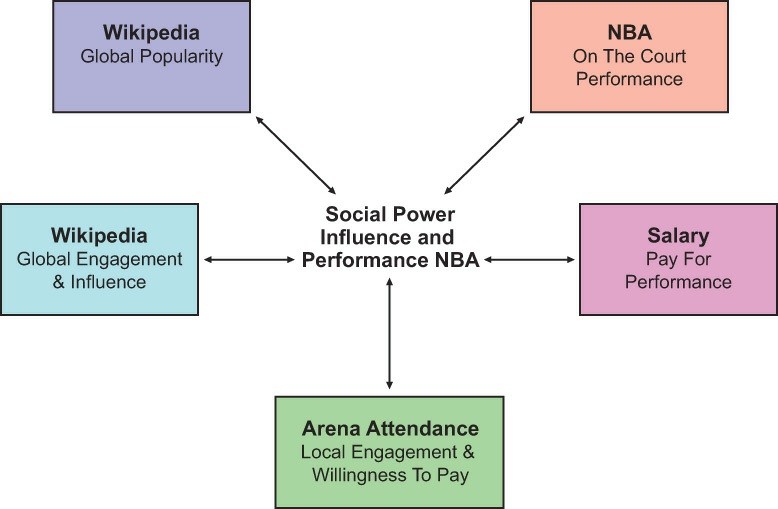
Does winning bring more fans to games?

What drives the valuation of teams more: attendance or the local real estate market?

To get the answers to these questions and others, data will need to be collected. As previously discussed, the 80/20 rule applies here. Eighty percent of this problem is collecting the data and then transforming the data. The other 20 percent is ML- and data science–related tasks like finding the right model, doing EDA, and feature engineering.

## Gathering the Data

In [Figure 6.1](#_bookmark0), there is a list of data sources to extract and transform.



**Figure 6.1** NBA Social Power Data Sources

Gathering this data represents a nontrivial software engineering problem. There are many obstacles to overcome, such as finding a good data source, writing code to extract it, abiding by the limitations of the API, and finally getting the data into the correct shape. The first step to collecting all of the data is to figure out which data source to collect first, and where to get it.

Knowing that the ultimate goal is to compare the social-media influence and power of NBA players, a great place to start is with the roster of the NBA players in the 2016–2017 season. In theory, this would be an easy task, but there are a few traps to collecting NBA data. The intuitive place to start would be to go to the official web site at nba.com. For some reason, however, many sports leagues make it difficult to download raw data from their sites. The NBA is no exception, and grabbing stats from their official web site is doable but challenging.

This brings up an interesting point about how to collect data. Often it is easy to collect data manually, that is, downloading from a web site and cleaning it up manually in Excel, Jupyter Notebook, or RStudio. This can be a very reasonable way to get started with a data science problem. If collecting one data source and cleaning it starts to take a few hours, however, it is probably best to look at writing code to solve the problem. There is no hard and fast rule, but experienced people figure out how to continuously make progress on a problem without getting blocked.

### Collecting the First Data Sources

Instead of starting with a thorny data source such as the official NBA web site, which actively prevents you from downloading its data, we are going to start with something relatively easy. To collect a first data source from basketball, you can download it directly from this book’s GitHub project (<https://github.com/noahgift/pragmaticai>) or from Basketball

Reference (<https://www.basketball-reference.com/leagues/NBA_2017_per_game.html>).

Doing ML in the real world is beyond just finding the right model for clean data; it means understanding how to set up your local environment as well.

To start running the code, a few steps are needed.

1. Create a virtual environment (based on Python 3.6).
2. Install a few packages that we will use for this chapter: i.e., Pandas, Jupyter. 3. Run this all through a Makefile.

[Listing 6.1](#_bookmark1) shows a setup command that creates a virtual environment for Python 3.6 and installs the packages listed in the requirements.txt file in [Listing 6.2](#_bookmark2). This can be executed all at once with this one liner.

make setup && install

## Listing 6.1 Makefile Contents

**Click here to view code image**

setup:

install:

python3 -m venv ~/.pragai6

pip install -r requirements.txt

## Listing 6.2 requirements.txt

Contents pytest nbval ipython requests

python-twitter pandas

pylint sensible jupyter matplotlib seaborn statsmodels sklearn wikipedia spacy ggplot

### Note

Another handy trick in dealing with Python virtual environments is to create an alias in your

.bashrc or .zshrc file that automatically activates the environment and changes into the directory all in one operation. The way I typically do this is by adding this snippet.

**Click here to view code image**

alias pragai6top="cd ~/src/pragai/chapter6\ && source ~/. Pragai6 /bin/activate"

To work on this chapter’s project, type pragai6top into the shell, and you will cd into the

correct project checkout and start your virtual environment. This is the power of using shell aliases in action. There are other tools that automatically do this for you, like pipenv; it may be worth exploring them as well.

To inspect the data, start a Jupyter Notebook using the command: jupyter notebook. Running this will launch a web browser that will allow you to explore existing notebooks or create new ones. If you have checked out the source code for this book’s GitHub project, you will see a file named basketball\_reference.ipynb.

This is a simple, hello world–type notebook with the data loaded into it. Loading a data set into Jupyter Notebook, or in the case of R, RStudio, is often the most convenient way to do initial validation and exploration of a data set. [Listing 6.3](#_bookmark3) shows how you can also explore the data from a regular IPython shell in addition to or instead of Jupyter.

**Listing 6.3** Jupyter Notebook Basketball Reference Exploration

**Click here to view code image**

import pandas as pd

nba = pd.read\_csv("data/nba\_2017\_br.csv") nba.describe()

### Note

Another useful technique is to get in the habit of ensuring Jupyter Notebooks are runnable using the nbval plugin for pytest. You can add a Makefile command test that will run all of your notebooks by issuing

make test

You can see what that would look like in a Makefile in the snippet below.

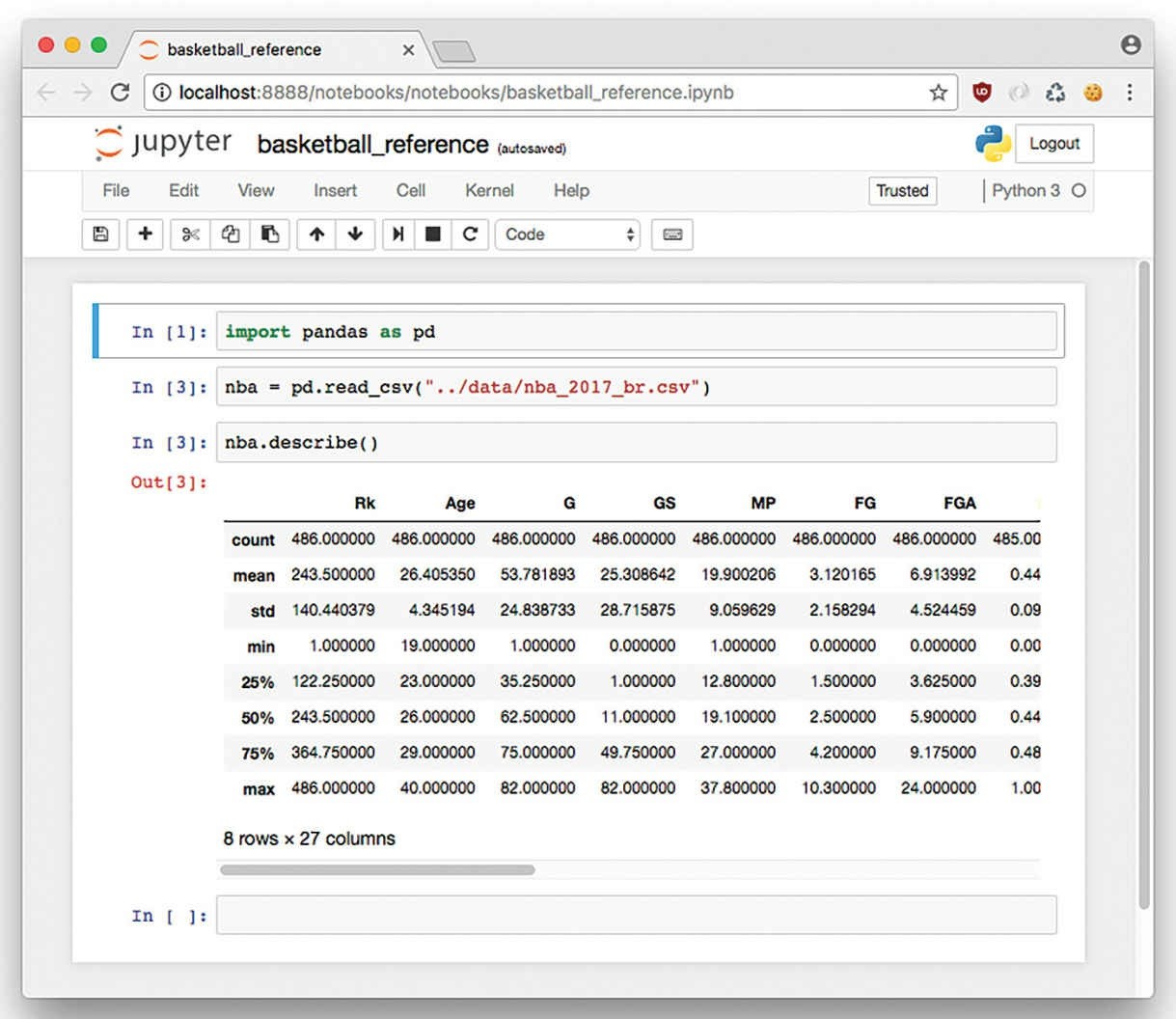
**Click here to view code image**

test:

py.test --nbval notebooks/\*.ipynb

Loading a CSV file into Pandas is easy if the CSV file has names for the columns and if the rows of each column are of equal length. If you are dealing with prepared data sets, then it is often if not always the case that the data will be in a suitable shape to load. In the real world, things are never this easy, and it is a battle to get the data into the correct shape as we will see later in this chapter.

[Figure 6.2](#_bookmark4) shows the output in Jupyter Notebook of the describe command. The describe function on a Pandas DataFrame provides descriptive statistics, including the number of columns, in this case 27, and median (this is the 50 percent row), for each column. At this point, it might be a good idea to play around with the Jupyter Notebook that was created and see what other insights you can observe. One of the things this data set doesn’t have, however, is a single metric to rank both offensive and defensive performance in a single statistic. To get this, we will need to combine this data set with other sources from ESPN and the NBA. This will raise the difficulty of the project significantly from simply using data to finding it, and then transforming it. One approach that is reasonable is to use a scraping tool like Scrapy, but in our situation, we can use a more ad hoc method. By going to the ESPN and NBA web sites, it is possible to cut and paste the data and put it into Excel. Then the data can be manually cleaned up and saved as a CSV file. For a small data set, this is often much quicker than trying to write a script to perform the same tasks.



**Figure 6.2** Basketball Reference DataFrame Describe Output Jupyter

Later, if this data needs to turn into a bigger project, this approach becomes a poor idea—but for prototyping, it is one of the strongest options. A key takeaway for messy data science problems is to continue to make forward progress without getting bogged down in too much detail. It is very easy to spend a lot of time automating a messy data source only to realize later that the signals are not helpful.

Grabbing the data from ESPN is a similar process as FiveThirtyEight, so I won’t describe how to collect it again. A couple of other data sources to collect are salary and endorsements. ESPN has the salary information, and Forbes has a small subset of the endorsement data for eight players. [Table 6.1](#_bookmark5) describes the shape of the data sources, summarizes their content, and defines their source. Mostly accomplished through manual work, there is a fairly impressive list of data sources.

### Table 6.1 NBA Data Sources

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Source** | **Filename** | **Rows** | **Summary** |
| Basketball Reference | nba\_2017\_attendance.csv | 30 | Stadium attendance |
| Forbes | nba\_2017\_endorsements.csv | 8 | Top players |
| Forbes | nba\_2017\_team\_valuations.csv | 30 | All teams |
| ESPN | nba\_2017\_salary.csv | 450 | Most players |
| NBA | nba\_2017\_pie.csv | 468 | All players |
| ESPN | nba\_2017\_real\_plus\_minus.csv | 468 | All players |
| Basketball Reference | nba\_2017\_br.csv | 468 | All players |
| FiveThirtyEight | nba\_2017\_elo.csv | 30 | Team rank |
| Basketball Reference | nba\_2017\_attendance.csv | 30 | Stadium attendance |
| Forbes | nba\_2017\_endorsements.csv | 8 | Top players |
| Forbes | nba\_2017\_team\_valuations.csv | 30 | All teams |
| ESPN | nba\_2017\_salary.csv | 450 | Most players |

There is still a lot of work left to get the rest of the data, mainly from Twitter and Wikipedia, and transform it into a unified data set. A couple of initially interesting possibilities are exploring the top eight player’s endorsements and exploring the valuation of the teams themselves.

**Exploring First Data Sources: Teams**

The first thing to do is to use a new Jupyter Notebook. In the GitHub repository, this has already been done for you, and it is called exploring\_team\_valuation\_nba. Next, import a common set of libraries that are typically used in exploring data in a Jupyter Notebook. This is shown in [Listing 6.4](#_bookmark6).

**Listing 6.4** Jupyter Notebook Common Initial Imports

**Click here to view code image**

import pandas as pd

import statsmodels.api as sm

import statsmodels.formula.api as smf import matplotlib.pyplot as plt import seaborn as sns

color = sns.color\_palette()

%matplotlib inline

Next, create a Pandas DataFrame for each source, as shown in [Listing 6.5](#_bookmark7).

**Listing 6.5** Create DataFrame for Sources

**Click here to view code image**

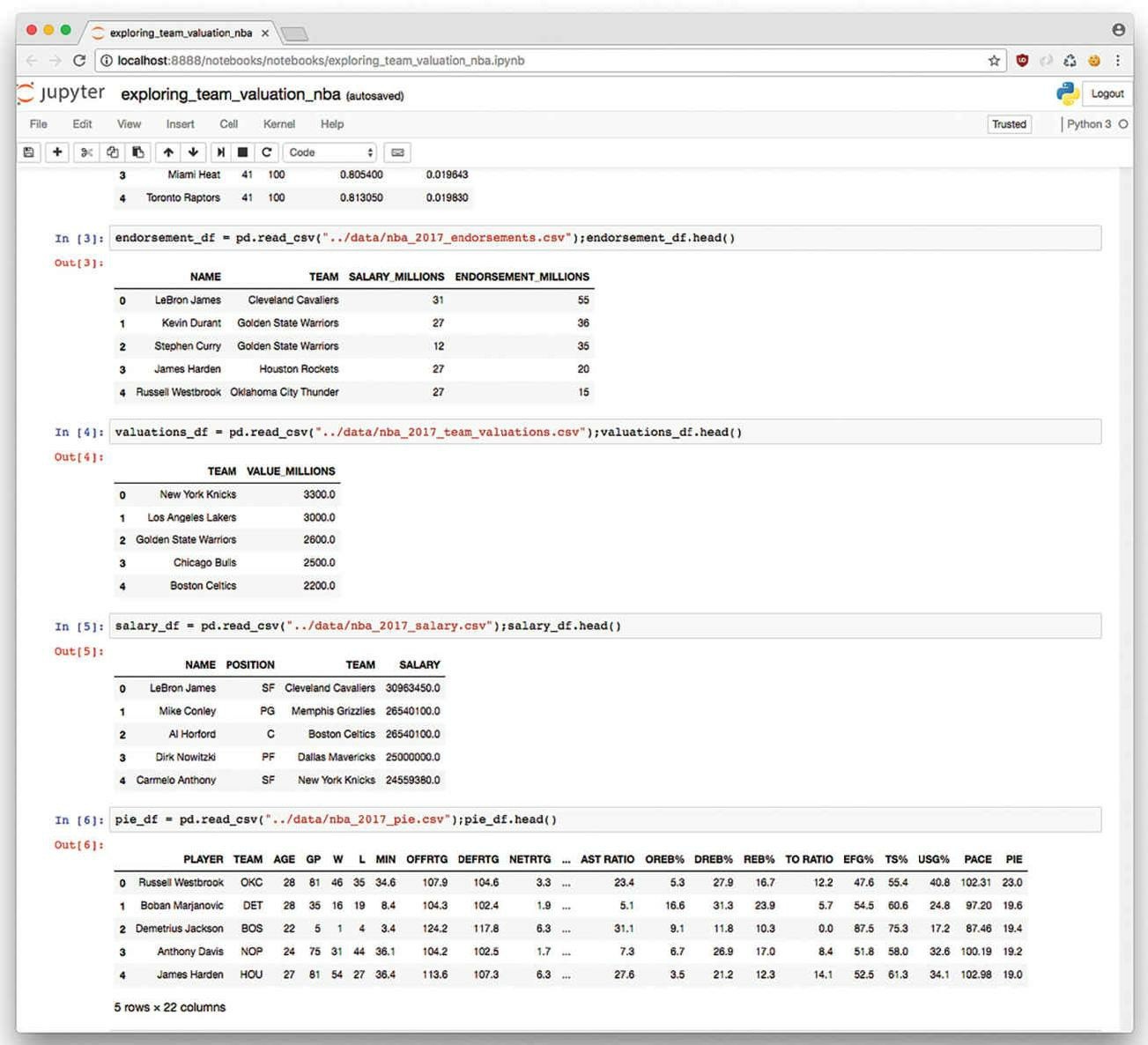
attendance\_df = pd.read\_csv("../data/nba\_2017\_attendance.csv") endorsement\_df = pd.read\_csv("../data/nba\_2017\_endorsements.csv") valuations\_df = pd.read\_csv("../data/nba\_2017\_team\_valuations.csv") salary\_df = pd.read\_csv("../data/nba\_2017\_salary.csv")

pie\_df = pd.read\_csv("../data/nba\_2017\_pie.csv")

plus\_minus\_df = pd.read\_csv("../data/nba\_2017\_real\_plus\_minus.csv") br\_stats\_df = pd.read\_csv("../data/nba\_2017\_br.csv")

elo\_df = pd.read\_csv("../data/nba\_2017\_elo.csv")

In [Figure 6.3](#_bookmark8), a chain of DataFrames are created—a common practice when collecting data in the wild.



**Figure 6.3** Multiple DataFrames Output in Jupyter

Here is a merge of attendance data with valuation data and a look at the first few rows.

**Click here to view code image**

In [14]: attendance\_valuation\_df =\ attendance\_df.merge(valuations\_df, how="inner", on="TEAM")

In [15]: attendance\_valuation\_df.head() Out[15]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEAM | GMS | PCT | TOTAL\_MILLIONS | AVG\_MILLIONS |
| 0 Chicago Bulls | 41 | 104 | 0.888882 | 0.021680 |
| 1 Dallas Mavericks | 41 | 103 | 0.811366 | 0.019789 |
| 2 Sacramento Kings | 41 | 101 | 0.721928 | 0.017608 |
| 3 Miami Heat | 41 | 100 | 0.805400 | 0.019643 |
| 4 Toronto Raptors | 41 | 100 | 0.813050 | 0.019830 |

Perform a pairplot using Seaborn, which is shown in [Figure 6.4](#_bookmark9).

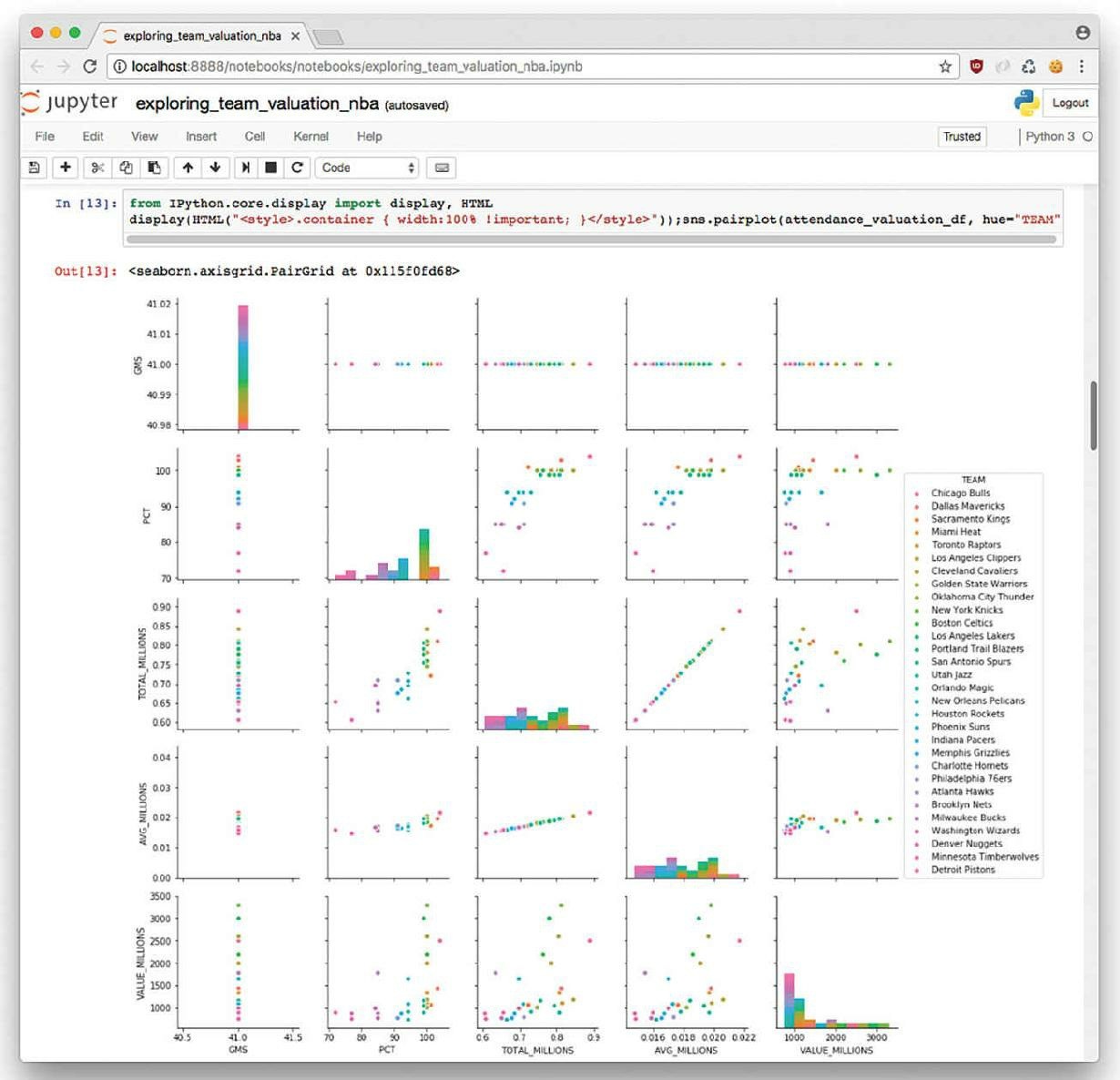
**Click here to view code image**

In [15]: from IPython.core.display import display, HTML

...: display(HTML("<style>.\

container{ width:100% !important; }</style>"));\ sns.pairplot(attendance\_valuation\_

...: df, hue="TEAM")



**Figure 6.4** Attendance/Valuation Pairplot

In looking at the plots there appears to be a relationship between attendance, either average or total, and the valuation of the team. Another way to dig deeper into this relationship is to create a correlation heatmap, shown in [Figure 6.5](#_bookmark10).

**Click here to view code image**

In [16]: corr = attendance\_valuation\_df.corr()

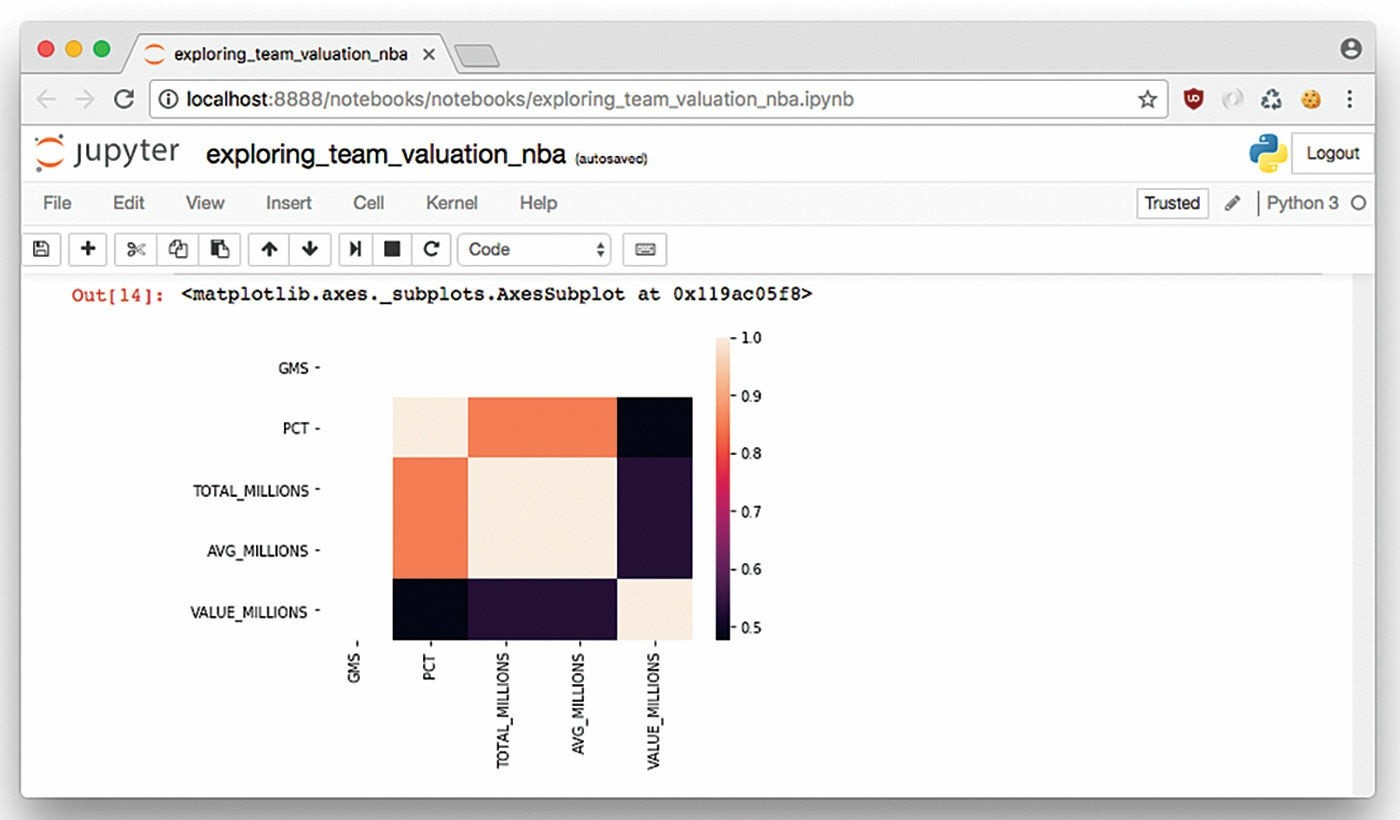
...: sns.heatmap(corr,

...: xticklabels=corr.columns.values,

...: yticklabels=corr.columns.values)

...:

Out[16]: <matplotlib.axes.\_subplots.AxesSubplot at 0x111007ac8>



**Figure 6.5** Attendance/Valuation Correlation Heatmap

The relationship visible in the pairplot is now more quantifiable. The heatmap shows a medium correlation between valuation and attendance, hovering around 50 percent. Another heatmap shows average attendance numbers versus valuation for every team in the NBA. To generate this type of heatmap in Seaborn, it is necessary to convert the data into a pivot table first. The plot can then be seen in [Figure 6.5](#_bookmark10).

**Click here to view code image**

In [18]: valuations = attendance\_valuation\_df.\ pivot("TEAM", "TOTAL\_MILLIONS", "VALUE\_MILLIONS")

In [19]: plt.subplots(figsize=(20,15))

...: ax = plt.axes()

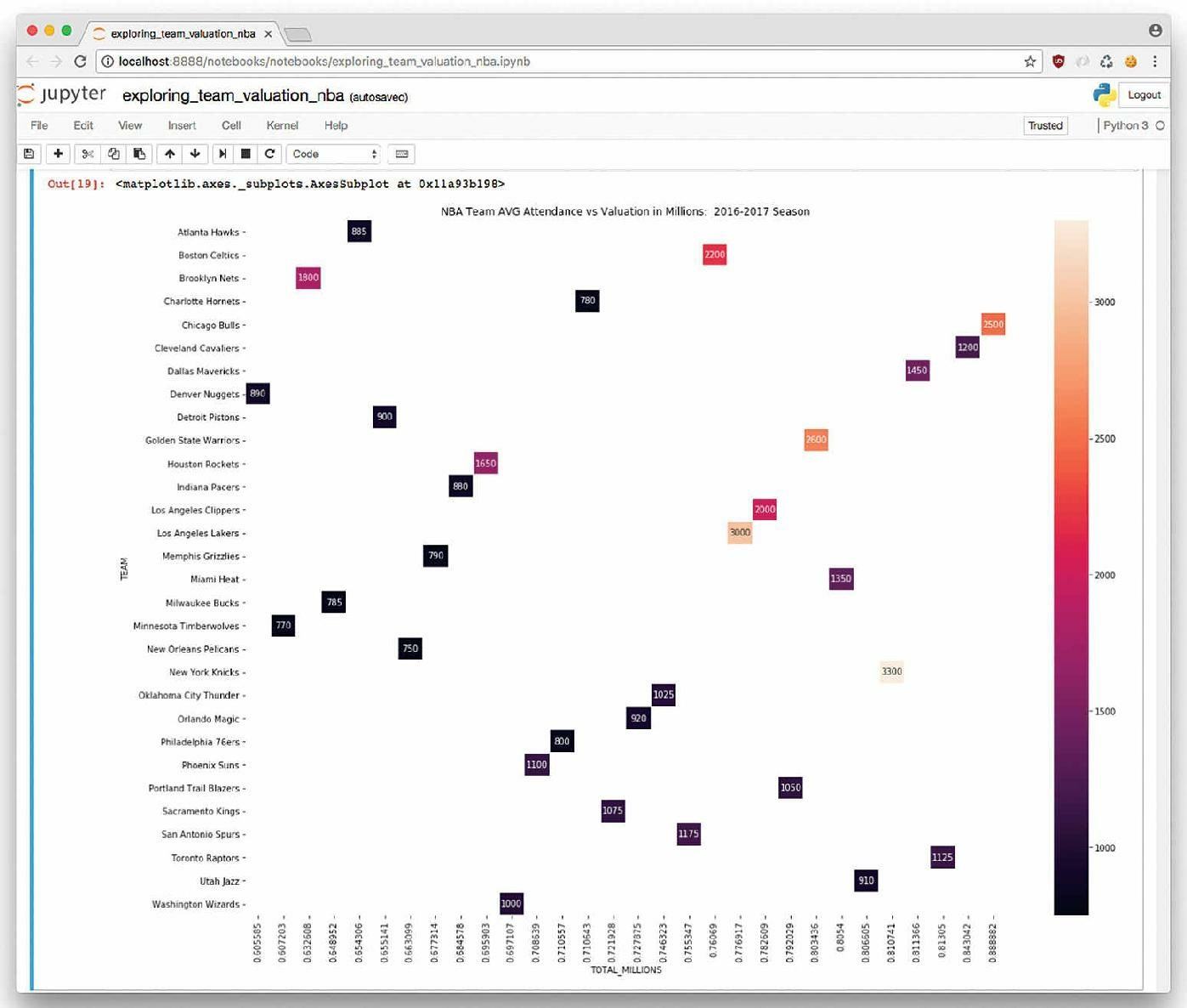
...: ax.set\_title("NBA Team AVG Attendance vs\ Valuation in Millions: 2016-2017 Season")

...: sns.heatmap(valuations,linewidths=.5, annot=True, fmt='g')

...:

Out[19]: <matplotlib.axes.\_subplots.AxesSubplot at 0x114d3d080>

In [Figure 6.6](#_bookmark11), a heatmap shows that there may be some interesting patterns to graph further, perhaps in a 3D plot. There are outliers in New York and Los Angles.



**Figure 6.6** NBA Teams Attendance versus Valuation Heatmap

### Exploring First Data Sources with Regression

[Figure 6.5](#_bookmark10) shows some fascinating outliers, for example, the Brooklyn Nets are valued at 1.8 billion dollars, yet they have one of the lowest attendance rates in the NBA. Something is going on here that is worth looking at. One way to further investigate is to use linear regression to try to explain the relationship. There are a few different ways to do this if you include both Python and R. In Python, two of the more common approaches are the StatsModels package and scikit-learn. Let’s explore both approaches.

With StatsModels, there is a great diagnostic output about performing a linear regression, and it has the feel of classic linear regression software like Minitab and R.

**Click here to view code image**

In [24]: results = smf.ols(

'VALUE\_MILLIONS ~TOTAL\_MILLIONS',

data=attendance\_valuation\_df).fit()

In [25]: print(results.summary())

OLS Regression Results

===============================================================

Dep. Variable: VALUE\_MILLIONS R-squared: 0.282

Model: OLS Adj. R-squared: 0.256

Method: Least Squares F-statistic: 10.98

Date: Thu, 10 Aug 2017 Prob (F-statistic):0.00255

Time: 14:21:16 Log-Likelihood: -234.04

No. Observations: 30 AIC: 472.1

Df Residuals: 28 BIC: 474.9

Df Model: 1

Covariance Type: nonrobust

================================================================

coef std err t P>|t|[0.025 0.975]

------------------------------------------------------------------

.....

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In looking at the results of the regression, it does appear that the variable TOTAL\_MILLIONS, which is total attendance in millions is statistically significant (measured in a *P* value of less than .05) in predicting changes in attendance. The R-squared value of .282 (or 28 percent) shows a “goodness of fit”; that is, how well the regression line perfectly fits the data.

Doing a bit more plotting and diagnostics will show how well this model is able to predict. Seaborn has a built in and very useful residplot that plots the residuals. This is shown in [Figure 6.7](#_bookmark12). Having randomly distributed residuals is the ideal scenario; if there are patterns in the plot, it could indicate issues with the model. In this case, there doesn’t seem to be a uniformly random pattern.

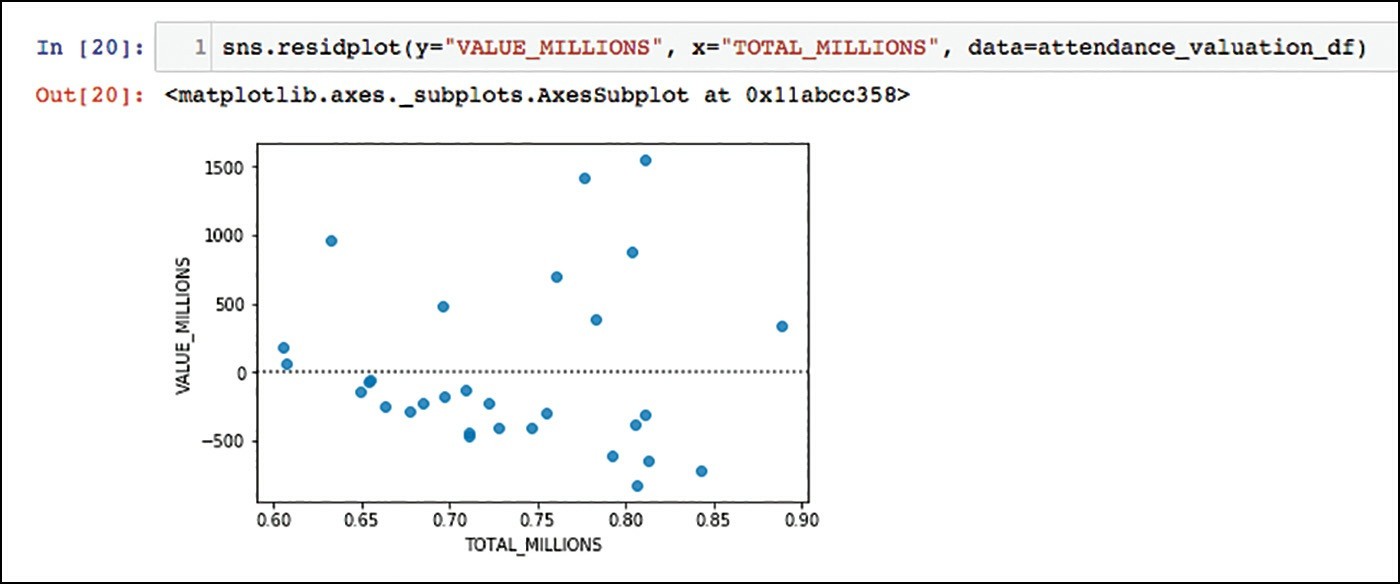
**Click here to view code image**

In [88]: sns.residplot(y="VALUE\_MILLIONS", x="TOTAL\_MILLIONS",

...: data=attendance\_valuation\_df)

...:

Out[88]: <matplotlib.axes.\_subplots.AxesSubplot at 0x114d3d080>



**Figure 6.7** NBA Teams Attendance versus Valuation Residual Plot

A common way to measure the accuracy of an ML or statistics prediction is to look at the root mean squared error (RMSE). Here is how to do it with the StatsModels.

**Click here to view code image**

In [92]: import statsmodels

...: rmse = statsmodels.tools.eval\_measures.rmse( attendance\_valuation\_predictions\_df["predicted"], attendance\_valuation\_predict

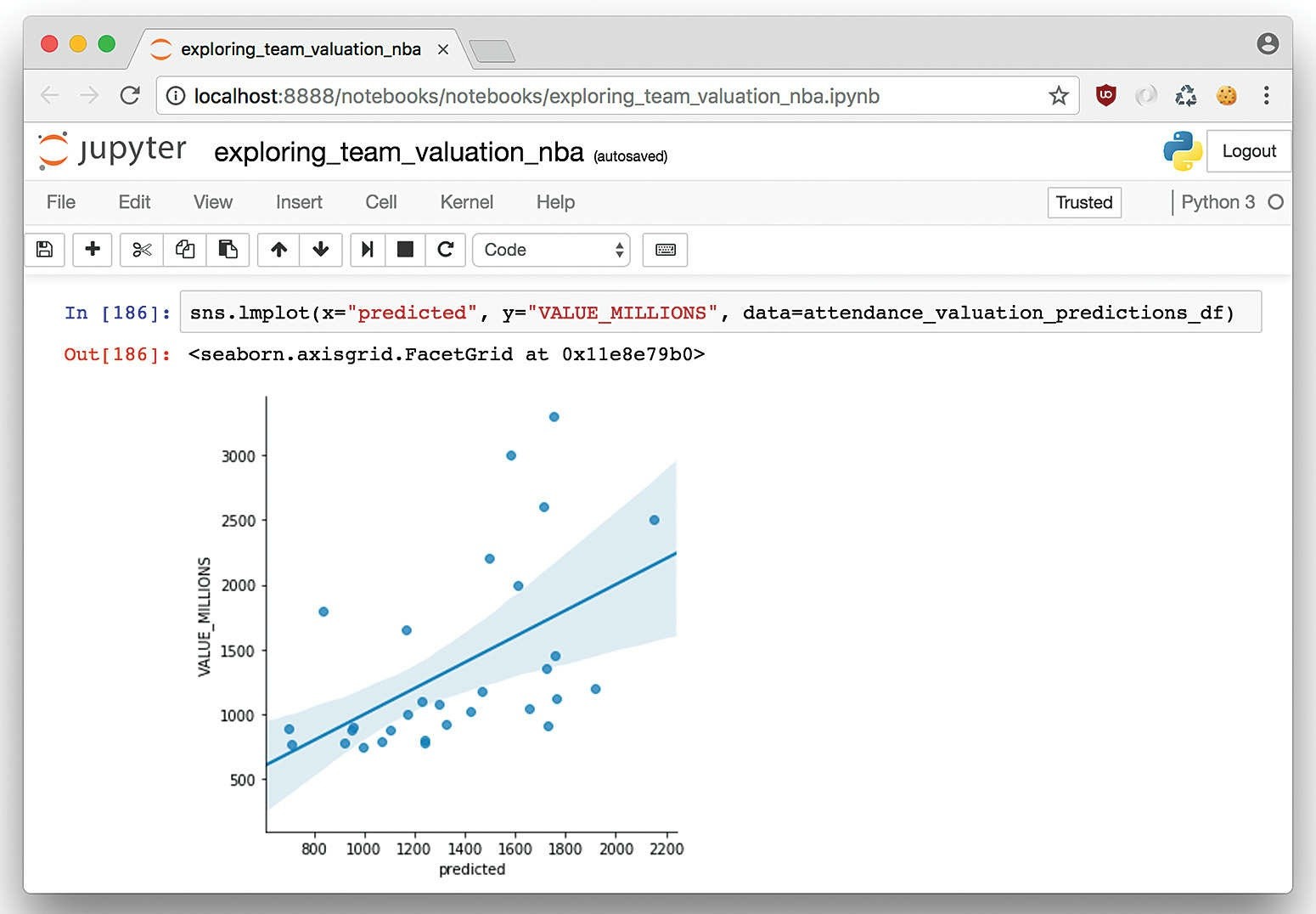
...: ions\_df["VALUE\_MILLIONS"])

...: rmse

...:

Out[92]: 591.33219017442696

The lower the RMSE, the better the prediction. To get a better prediction accuracy, we need to figure out a way to lower this RMSE. In addition, having a larger set of data such that the model could be split into test versus training data would ensure better accuracy and reduce the chance of overfitting. A further diagnostic step is to plot the predicted values of the linear regression versus the actual values. In [Figure 6.8](#_bookmark13), an lmplot of the predicted and actual is shown, and it is obvious that this isn’t that great a prediction model. It is a good start though, and often this is how ML models are created— by finding correlations and/or statistically significant relationships, then deciding it is worth the effort to collect more data.



**Figure 6.8** Predicted versus Actual Plot of Team Valuation

An initial conclusion is that while there is a relationship between attendance and valuation of an NBA team, there are missing or *latent variables*. An initial hunch is that population of the region, median real estate prices, and how good the team is (ELO ranking and winning percentage) all could play a role here.

**Click here to view code image**

In [89]: attendance\_valuation\_predictions\_df =\ attendance\_valuation\_df.copy()

In [90]: attendance\_valuation\_predictions\_df["predicted"] =\ results.predict()

In [91]: sns.lmplot(x="predicted", y="VALUE\_MILLIONS",\ data=attendance\_valuation\_predictions\_df)

Out[91]: <seaborn.axisgrid.FacetGrid at 0x1178d2198>

### Unsupervised Machine Learning: Clustering First Data Sources

A next step in learning more about NBA teams is to use unsupervised ML to cluster the data to find more insights. I was able to manually find median home price data for a county

on <https://www.zillow.com/research/>and the population for each county from the census on

<https://www.census.gov/data/tables/2016/demo/popest/counties-total.html>. All this new data can be loaded with a new DataFrame.

**Click here to view code image**

In [99]: val\_housing\_win\_df

= pd.read\_csv("../data/nba\_2017\_att\_val\_elo\_win\_housing.csv") In [100]: val\_housing\_win\_df.columns

Out[100]:

Index(['TEAM', 'GMS', 'PCT\_ATTENDANCE', 'WINNING\_SEASON', 'TOTAL\_ATTENDANCE\_MILLIONS', 'VALUE\_MILLIONS',

'ELO', 'CONF', 'COUNTY', 'MEDIAN\_HOME\_PRICE\_COUNTY\_MILLONS', 'COUNTY\_POPULATION\_MILLIONS'],

dtype='object')

* 1. earest neighbors (kNN) clustering works by determining the Euclidean distance between points. Attributes being clustered needed to be scaled so one attribute doesn’t have a different scale than another, which would distort the clustering. In addition, clustering is more art than science, and picking the correct number of clusters can be a trial-and-error process. Here is how scaling works in practice.

**Click here to view code image**

In [102]: numerical\_df = val\_housing\_win\_df.loc[:,\ ["TOTAL\_ATTENDANCE\_MILLIONS", "ELO", "VALUE\_MILLIONS", "MEDIAN\_HOME\_PRICE\_COUNT

...: Y\_MILLONS"]]

In [103]: from sklearn.preprocessing import MinMaxScaler

...: scaler = MinMaxScaler()

...: print(scaler.fit(numerical\_df))

...: print(scaler.transform(numerical\_df)) MinMaxScaler(copy=True, feature\_range=(0, 1))

|  |  |  |  |
| --- | --- | --- | --- |
| [[ 1. | 0.41898148 | 0.68627451 | 0.08776879] |
| [ 0.72637903 | 0.18981481 | 0.2745098 | 0.11603661] |
| [ 0.41067502 | 0.12731481 | 0.12745098 | 0.13419221]… |

In this example, MinMaxScaler is being used from scikit-learn. It converts all numerical values to a value between 0 and 1. Next, sklearn.cluster is performed against the scaled data, and then the cluster results are attached to a new column.

**Click here to view code image**

In [104]: from sklearn.cluster import KMeans

...: k\_means = KMeans(n\_clusters=3)

...: kmeans = k\_means.fit(scaler.transform(numerical\_df))

...: val\_housing\_win\_df['cluster'] = kmeans.labels\_

...: val\_housing\_win\_df.head()

...:

Out[104]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEAM | GMS | PCT\_ATTENDANCE | WINNING\_SEASON | \ |
| 0 Chicago Bulls | 41 | 104 | 1 |  |
| 1 Dallas Mavericks | 41 | 103 | 0 |  |
| 2 Sacramento Kings | 41 | 101 | 0 |  |
| 3 Miami Heat | 41 | 100 | 1 |  |
| 4 Toronto Raptors | 41 | 100 | 1 |  |

TOTAL\_ATTENDANCE\_MILLIONS VALUE\_MILLIONS ELO CONF 0 0.888882 2500 1519 East

1 0.811366 1450 1420 West

2 0.721928 1075 1393 West

3 0.805400 1350 1569 East

4 0.813050 1125 1600 East MEDIAN\_HOME\_PRICE\_COUNTY\_MILLONS cluster

0 269900.0 1

1 314990.0 1

2 343950.0 0

3 389000.0 1

4 390000.0 1

At this point, there is enough of a solution to provide instant value to a company, and the beginning of a data pipeline is forming. Next let’s use R and ggplot to plot the clusters. In order to bring this data set into R, we can write this out to a CSV file.

**Click here to view code image**

In [105]: val\_housing\_win\_df.to\_csv( "../data/nba\_2017\_att\_val\_elo\_win\_housing\_cluster.csv"

)

### Plotting kNN Clustering in 3D with R

A highlight of the R language is the ability to create advanced plots with meaningful text. Being capable of coding solutions in R and Python opens up a wider variety of solutions in ML. In this particular situation, we are going to use the R 3D scatter plot library along with RStudio to make a sophisticated plot of the relationships we have learned about using kNN cluster. In the GitHub project for this chapter, there is R markdown notebook that has the code and plot; you can also follow along by using the preview function in RStudio for notebooks.

To get started in the console in RStudio (or an R shell), import the scatterplot3d library and load the data using the following commands.

**Click here to view code image**

* library("scatterplot3d",

lib.loc="/Library/Frameworks/R.framework/\ Versions/3.4/Resources/library")

* team\_cluster <- read\_csv("~/src/aibook/src/chapter7/data/\ nba\_2017\_att\_val\_elo\_win\_housing\_cluster.csv",

+ col\_types = cols(X1 = col\_skip()))

Next, a function is created to convert the data types into a format that the scatterplot3d library is expecting.

**Click here to view code image**

* cluster\_to\_numeric <- function(column){

+ converted\_column <- as.numeric(unlist(column))

+ return(converted\_column)

+ }

A new column is created to hold color data about each cluster.

**Click here to view code image**

* team\_cluster$pcolor[team\_cluster$cluster == 0] <- "red"
* team\_cluster$pcolor[team\_cluster$cluster == 1] <- "blue"
* team\_cluster$pcolor[team\_cluster$cluster == 2] <- "darkgreen"

A skeleton 3D plot is created.

**Click here to view code image**

* s3d <- scatterplot3d(

+ cluster\_to\_numeric(team\_cluster["VALUE\_MILLIONS"]),

+ cluster\_to\_numeric(

team\_cluster["MEDIAN\_HOME\_PRICE\_COUNTY\_MILLIONS"]),

+ cluster\_to\_numeric(team\_cluster["ELO"]),

+ color = team\_cluster$pcolor,

+ pch=19,

+ type="h",

+ lty.hplot=2,

+ main="3-D Scatterplot NBA Teams 2016-2017: Value, Performance, Home Prices with kNN Clustering",

+ zlab="Team Performance (ELO)",

+ xlab="Value of Team in Millions",

+ ylab="Median Home Price County Millions"

+ )

>

To plot the text in the correct location on the 3D space requires a little bit of work.

**Click here to view code image**

s3d.coords <- s3d$xyz.convert( cluster\_to\_numeric(team\_cluster["VALUE\_MILLIONS"]),

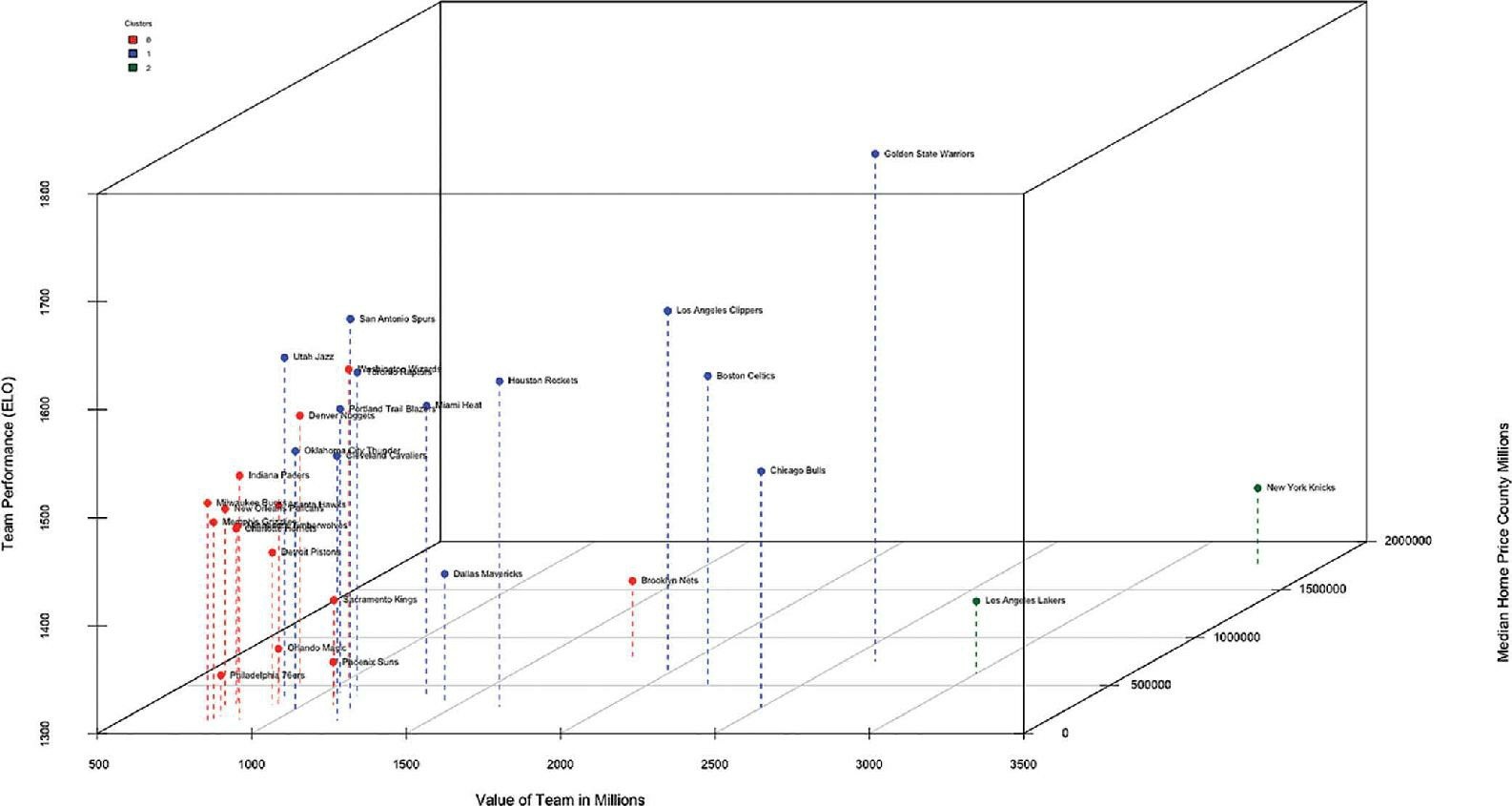
cluster\_to\_numeric( team\_cluster["MEDIAN\_HOME\_PRICE\_COUNTY\_MILLIONS"]),

cluster\_to\_numeric(team\_cluster["ELO"]))

#plot text

text(s3d.coords$x, s3d.coords$y, # x and y coordinates labels=team\_cluster$TEAM, # text to plot pos=4, cex=.6) # shrink text)

The plot shown in [Figure 6.9](#_bookmark14) shows some unusual patterns. The New York Knicks and the Los Angeles Lakers are two of the worst teams in basketball, yet are the most valuable. In addition, you can see that they are in cities that have some of the highest median home prices, which is playing a role in their high valuation. As a result of all of this, they are in their own cluster.



**Figure 6.9** 3D Scatter Plot of NBA Teams: 2016-2017 with kNN

The blue cluster is mostly a collection of the best teams in the NBA. They also tend to be in cities with higher median home prices but a wide variation of actual value. This makes me suspect that real estate plays a bigger role in team valuation than actual performance (which lines up with previous linear regressions).

The red cluster shows teams that are generally below average in performance, have below-average valuation, and have below-average real estate prices. The exception is the Brooklyn Nets, which is on its way to being a Los Angeles Lakers– and New York Knicks–type team: low performing, yet highly valued.

R has yet one more way to visualize these relationships in multiple dimensions. Next, we are going to create a plot using ggplot in R.

The first thing to do in plotting the relationship in the new graph is to make a logical name for the clusters. The 3D plot gave us some great ideas about how to name clusters. Cluster 0 appears to be a low valuation/low performance cluster, Cluster 1 is a medium valuation/high performance cluster, and Cluster 2 is a high valuation/low performance cluster. One note to add is that cluster number selection is a complex subject. (See Appendix B for more information on the topic.)

**Click here to view code image**

* team\_cluster <- read\_csv("nba\_cluster.csv",

+ col\_types = cols(X1 = col\_skip()))

* library("ggplot2")

>

* #Name Clusters
* team\_cluster$cluster\_name[team\_cluster$cluster == 0] <- "Low" Unknown or uninitialised column: 'cluster\_name'.
* team\_cluster$cluster\_name[team\_cluster$

cluster == 1] <- "Medium Valuation/High Performance"

* team\_cluster$cluster\_name[team\_cluster$

cluster == 2] <- "High Valuation/Low Performance"

Next, we can use these cluster names to facet (create multiple plots in each plot). In addition, ggplot has the ability to create many other dimensions, and we are going to use them all: color to show winning team percentages and losing team percentages, size to show the differences in median home prices in the county, and the shape to represent the Eastern or Western Conference of the NBA.

**Click here to view code image**

* p <- ggplot(data = team\_cluster) +

+ geom\_point(mapping = aes(x = ELO,

+ y = VALUE\_MILLIONS,

+ color =

factor(WINNING\_SEASON, labels= c("LOSING","WINNING")),

+size = MEDIAN\_HOME\_PRICE\_COUNTY\_MILLIONS,

+ shape = CONF)) +

+ facet\_wrap(~ cluster\_name) +

+ ggtitle("NBA Teams 2016-2017 Faceted Plot") +

+ ylab("Value NBA Team in Millions") +

+ xlab("Relative Team Performance (ELO)") +

+ geom\_text(aes(x = ELO, y = VALUE\_MILLIONS,

+ label=ifelse(VALUE\_MILLIONS>1200,

+ as.character(TEAM),'')),hjust=.35,vjust=1)

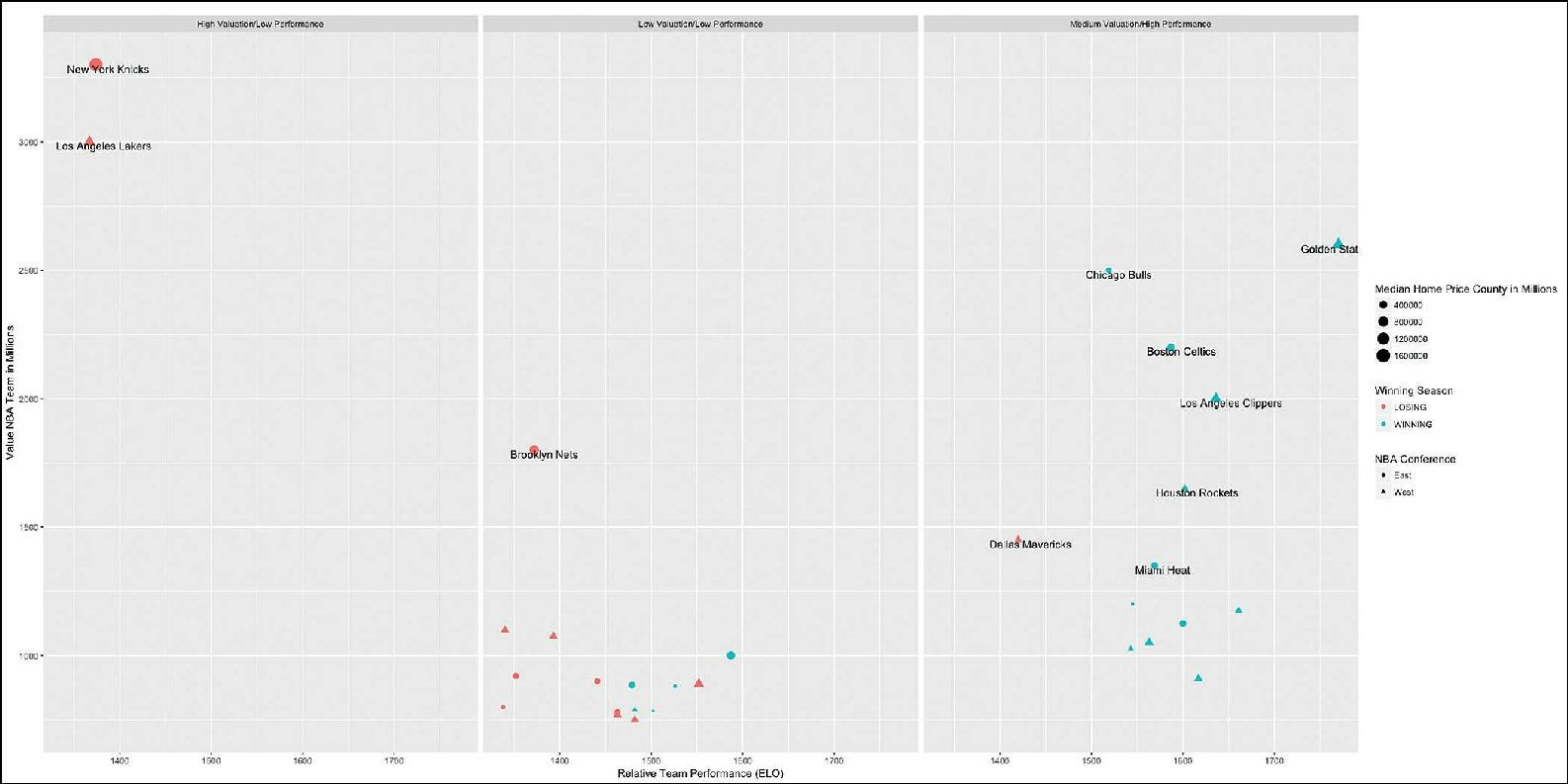
Notice that geom\_text only prints the name of the team if the valuation is over 1200. This allows the plot to be more readable and not overwhelmed with overlapping text. In the final snippet, the legend titles are changed. Note also the color is changed to be a factor with one of two values, versus the default of 0, .25, .50, 1. The output of the plot appears in [Figure 6.10](#_bookmark15). The faceting feature of ggplot really shows how clustering has added value to the exploration of data. Using R to do advanced plotting is a great idea even if you are an expert at another ML language like Python or Scala. The results speak for themselves.

**Click here to view code image**

#Change legends p +

guides(color = guide\_legend(title = "Winning Season")) + guides(size = guide\_legend(

+ title = "Median Home Price County in Millions" )) + guides(shape = guide\_legend(title = "NBA Conference"))



**Figure 6.10** ggplot Faceted Plot of NBA Teams: 2016-2017 with kNN

# Collecting Challenging Data Sources

With a good set of data around teams already collected, it is time to get into more challenging data sources. This is where things start to get more real. There are some huge issues with collecting random data sources: API limits, undocumented APIs, dirty data, and more.

## Col ecting Wikipedia Pageviews for Athletes

Here are a few of the problems to solve.

* + 1. How to reverse engineer the Wikipedia system to get pageviews (or find hidden API documentation)
    2. How to find a way to generate Wikipedia handles (they may not be the same name as their NBA name)
    3. How to join the DataFrame with the rest of the data

Here is how to accomplish this in Python. The entire source for this example is in the GitHub repo for the book, but it will be analyzed in these sections. Below is the example URL for Wikipedia pageviews and the four modules needed. The requests library will make the HTTP calls, Pandas will convert the results into a DataFrame, and the Wikipedia library will be used for a heuristic around detecting the proper Wikipedia URL for an athlete.

**Click here to view code image**

"""

Example Route To Construct:

https://wikimedia.org/api/rest\_v1/ + metrics/pageviews/per-article/ + en.wikipedia/all-access/user/ + LeBron\_James/daily/2015070100/2017070500 +

"""

import requests import pandas as pd import time

import wikipedia

BASE\_URL =\

"https://wikimedia.org/api/rest\_v1/\ metrics/pageviews/per-article/en.wikipedia/all-access/user"

Next, the following code constructs a URL that has the data range and username.

**Click here to view code image**

def construct\_url(handle, period, start, end): """Constructs a URL based on arguments

Should construct the following URL:

/LeBron\_James/daily/2015070100/2017070500 """

urls = [BASE\_URL, handle, period, start, end] constructed = str.join('/', urls)

return constructed

def query\_wikipedia\_pageviews(url): res = requests.get(url)

return res.json()

def wikipedia\_pageviews(handle, period, start, end): """Returns JSON"""

constructed\_url = construct\_url(handle, period, start,end) pageviews = query\_wikipedia\_pageviews(url=constructed\_url) return pageviews

The following function automatically populates a query for 2016. This could later be made more abstract, but for now, this is “hacker” code where hard coding things for speed may be worth the technical debt. Notice as well that a sleep is set to 0 but may need to be enabled if we hit API limits. This is a common pattern when first hitting APIs; they could behave in unexpected ways, so sleeping at some interval can often work around this issue, again, as a temporary hack.

**Click here to view code image**

def wikipedia\_2016(handle,sleep=0): """Retrieve pageviews for 2016"""

print("SLEEP: {sleep}".format(sleep=sleep)) time.sleep(sleep)

pageviews = wikipedia\_pageviews(handle=handle, period="daily", start="2016010100", end="2016123100")

if not 'items' in pageviews:

print("NO PAGEVIEWS: {handle}".format(handle=handle))

return None return pageviews

Next, the results are converted into a Pandas DataFrame.

**Click here to view code image**

def create\_wikipedia\_df(handles): """Creates a Dataframe of Pageviews"""

pageviews = [] timestamps = [] names = []

wikipedia\_handles = []

for name, handle in handles.items(): pageviews\_record = wikipedia\_2016(handle) if pageviews\_record is None:

continue

for record in pageviews\_record['items']: pageviews.append(record['views']) timestamps.append(record['timestamp']) names.append(name) wikipedia\_handles.append(handle)

data = {

"names": names,

"wikipedia\_handles": wikipedia\_handles, "pageviews": pageviews,

"timestamps": timestamps

}

df = pd.DataFrame(data) return df

A trickier section of the code begins here because some heuristics are needed to guess the right handle. For a first pass, a guess is made that most handles are simply first\_last. A second pass appends “(basketball)” to the name, which is a common Wikipedia strategy for disambiguation.

**Click here to view code image**

def create\_wikipedia\_handle(raw\_handle):

"""Takes a raw handle and converts it to a wikipedia handle"""

wikipedia\_handle = raw\_handle.replace(" ", "\_") return wikipedia\_handle

def create\_wikipedia\_nba\_handle(name): """Appends basketball to link"""

url = " ".join([name, "(basketball)"]) return url

def wikipedia\_current\_nba\_roster():

"""Gets all links on wikipedia current roster page"""

links = {}

nba = wikipedia.page("List\_of\_current\_NBA\_team\_rosters") for link in nba.links:

links[link] = create\_wikipedia\_handle(link) return links

This code runs both heuristics and returns verified handles and guesses.

**Click here to view code image**

def guess\_wikipedia\_nba\_handle(data="data/nba\_2017\_br.csv"): """Attempt to get the correct wikipedia handle"""

links = wikipedia\_current\_nba\_roster() nba = pd.read\_csv(data)

count = 0 verified = {} guesses = {}

for player in nba["Player"].values: if player in links:

print("Player: {player}, Link: {link} ".\ format(player=player,

link=links[player])) print(count)

count += 1

verified[player] = links[player] #add wikipedia link else:

print("NO MATCH: {player}".format(player=player)) guesses[player] = create\_wikipedia\_handle(player)

return verified, guesses

Next, the Wikipedia Python library is used to convert failed initial guesses of first and last name and looks for “NBA” in the page summary. This is another decent hack to get a few more matches.

**Click here to view code image**

def validate\_wikipedia\_guesses(guesses): """Validate guessed wikipedia accounts"""

verified = {} wrong = {}

for name, link in guesses.items(): try:

page = wikipedia.page(link) except (wikipedia.DisambiguationError, wikipedia.PageError) as error:

#try basketball suffix

nba\_handle = create\_wikipedia\_nba\_handle(name) try:

page = wikipedia.page(nba\_handle) print("Initial wikipedia URL Failed:\

{error}".format(error=error)) except (wikipedia.DisambiguationError,

wikipedia.PageError) as error:

print("Second Match Failure: {error}".\ format(error=error))

wrong[name] = link continue

if "NBA" in page.summary: verified[name] = link

else:

print("NO GUESS MATCH: {name}".format(name=name)) wrong[name] = link

return verified, wrong

At the end of the script, everything is run and the output is used to create a new CSV file.

**Click here to view code image**

def clean\_wikipedia\_handles(data="data/nba\_2017\_br.csv"): """Clean Handles"""

verified, guesses = guess\_wikipedia\_nba\_handle(data=data) verified\_cleaned, wrong = validate\_wikipedia\_guesses(guesses) print("WRONG Matches: {wrong}".format(wrong=wrong))

handles = {\*\*verified, \*\*verified\_cleaned} return handles

def nba\_wikipedia\_dataframe(data="data/nba\_2017\_br.csv"): handles = clean\_wikipedia\_handles(data=data)

df = create\_wikipedia\_df(handles) return df

def create\_wikipedia\_csv(data="data/nba\_2017\_br.csv"): df = nba\_wikipedia\_dataframe(data=data) df.to\_csv("data/wikipedia\_nba.csv")

if name == " main ": create\_wikipedia\_csv()

All together, something like this can take anywhere from a few hours to a few days and represents the realism of slogging through random data sources to solve a problem.

## Col ecting Twitter Engagement for Athletes

Collection of data from Twitter has elements that are a bit easier. For one thing, there is a great library in Python, aptly named twitter. There are still some challenges as well, however. Here they are laid out.

1. Summarizing engagement using descriptive statistics
2. Finding the right Twitter handles (handle names on Twitter are even harder to find than on Wikipedia)
3. Joining the DataFrame with the rest of the data

First, create a config file config.py and put credentials for the Twitter API inside of it. Then the

.import config will create a namespace to use these credentials. Also, Twitter error handling is imported as well as Pandas and NumPy.

**Click here to view code image**

import time

import twitter

from . import config import pandas as pd import numpy as np

from twitter.error import TwitterError

The following code talks to Twitter and grabs 200 tweets and converts them into a Pandas DataFrame. Note how this pattern is used frequently in talking with APIs; the columns are put into a list, then the list of columns is used to create a DataFrame.

**Click here to view code image**

def api\_handler():

"""Creates connection to Twitter API"""

api = twitter.Api(consumer\_key=config.CONSUMER\_KEY,

consumer\_secret=config.CONSUMER\_SECRET, access\_token\_key=config.ACCESS\_TOKEN\_KEY, access\_token\_secret=config.ACCESS\_TOKEN\_SECRET) return api

def tweets\_by\_user(api, user, count=200):

"""Grabs the "n" number of tweets. Defaults to 200"""

tweets = api.GetUserTimeline(screen\_name=user, count=count) return tweets

def stats\_to\_df(tweets):

"""Takes twitter stats and converts them to a dataframe"""

records = []

for tweet in tweets: records.append({"created\_at":tweet.created\_at,

"screen\_name":tweet.user.screen\_name, "retweet\_count":tweet.retweet\_count, "favorite\_count":tweet.favorite\_count})

df = pd.DataFrame(data=records) return df

def stats\_df(user):

"""Returns a dataframe of stats"""

api = api\_handler()

tweets = tweets\_by\_user(api, user) df = stats\_to\_df(tweets)

return df

The last function stats\_df, can now be used to interactively explore the results of a Twitter API call. Here is an example of LeBron James’ descriptive statistics.

**Click here to view code image**

df = stats\_df(user="KingJames") In [34]: df.describe() Out[34]:

|  |  |  |
| --- | --- | --- |
|  | favorite\_count | retweet\_count |
| count | 200.000000 | 200.000000 |
| mean | 11680.670000 | 4970.585000 |
| std | 20694.982228 | 9230.301069 |
| min | 0.000000 | 39.000000 |
| 25% | 1589.500000 | 419.750000 |
| 50% | 4659.500000 | 1157.500000 |
| 75% | 13217.750000 | 4881.000000 |
| max | 128614.000000 | 70601.000000 |

In [35]: df.corr() Out[35]:

|  |  |  |
| --- | --- | --- |
|  | favorite\_count | retweet\_count |
| favorite\_count | 1.000000 | 0.904623 |
| retweet\_count | 0.904623 | 1.000000 |

In the following code, the Twitter API is called with a slight sleep to avoid running into API throttling. Notice that the Twitter handles are being pulled from a CSV file. Basketball Reference also keeps a large selection of Twitter accounts. Another option would have been to find them manually.

**Click here to view code image**

def twitter\_handles(sleep=.5,data="data/twitter\_nba\_combined.csv"): """yield handles"""

nba = pd.read\_csv(data)

for handle in nba["twitter\_handle"]: time.sleep(sleep) #Avoid throttling in twitter api try:

df = stats\_df(handle) except TwitterError as error:

print("Error {handle} and error msg {error}".format( handle=handle,error=error))

df = None yield df

def median\_engagement(data="data/twitter\_nba\_combined.csv"): """Median engagement on twitter"""

favorite\_count = [] retweet\_count = []

nba = pd.read\_csv(data)

for record in twitter\_handles(data=data): print(record)

#None records stored as Nan value if record is None:

print("NO RECORD: {record}".format(record=record)) favorite\_count.append(np.nan) retweet\_count.append(np.nan)

continue try:

favorite\_count.append(record['favorite\_count'].median()) retweet\_count.append(record["retweet\_count"].median())

except KeyError as error:

print("No values found to append {error}".\ format(error=error))

favorite\_count.append(np.nan) retweet\_count.append(np.nan)

print("Creating DF") nba['twitter\_favorite\_count'] = favorite\_count nba['twitter\_retweet\_count'] = retweet\_count return nba

At the end of all of this, a new CSV file is created.

**Click here to view code image**

def create\_twitter\_csv(data="data/nba\_2016\_2017\_wikipedia.csv"): nba = median\_engagement(data) nba.to\_csv("data/nba\_2016\_2017\_wikipedia\_twitter.csv")

## Exploring NBA Athlete Data

To explore the athlete data, a new Jupyter Notebook will be created. This notebook is called nba\_player\_power\_influence\_performance. To begin, import a few libraries that are commonly used.

**Click here to view code image**

In [106]: import pandas as pd

...: import numpy as np

...: import statsmodels.api as sm

...: import statsmodels.formula.api as smf

...: import matplotlib.pyplot as plt

...: import seaborn as sns

...: from sklearn.cluster import KMeans

...: color = sns.color\_palette()

...: from IPython.core.display import display, HTML

...: display(HTML("<style>.container\

{ width:100% !important; }</style>"))

...: %matplotlib inline

...:

<IPython.core.display.HTML object>

Next, load the data files in the project and rename the columns.

**Click here to view code image**

In [108]: attendance\_valuation\_elo\_df =\ pd.read\_csv("../data/nba\_2017\_att\_val\_elo.csv")

In [109]: salary\_df = pd.read\_csv("../data/nba\_2017\_salary.csv") In [110]: pie\_df = pd.read\_csv("../data/nba\_2017\_pie.csv")

In [111]: plus\_minus\_df =\ pd.read\_csv("../data/nba\_2017\_real\_plus\_minus.csv")

In [112]: br\_stats\_df = pd.read\_csv("../data/nba\_2017\_br.csv") In [113]: plus\_minus\_df.rename(

columns={"NAME":"PLAYER", "WINS": "WINS\_RPM"}, inplace=True)

...: players = []

...: for player in plus\_minus\_df["PLAYER"]:

...: plyr, \_ = player.split(",")

...: players.append(plyr)

...: plus\_minus\_df.drop(["PLAYER"], inplace=True, axis=1)

...: plus\_minus\_df["PLAYER"] = players

...:

There are some duplicate sources, so these can also be dropped.

**Click here to view code image**

In [114]: nba\_players\_df = br\_stats\_df.copy()

...: nba\_players\_df.rename(

columns={'Player': 'PLAYER','Pos':'POSITION', 'Tm': "TEAM", 'Age': 'AGE', "PS/G": "POINTS"}, i

...: nplace=True)

...: nba\_players\_df.drop(["G", "GS", "TEAM"], inplace=True, axis=1)

...: nba\_players\_df =\ nba\_players\_df.merge(plus\_minus\_df, how="inner", on="PLAYER")

...:

In [115]: pie\_df\_subset = pie\_df[["PLAYER", "PIE", "PACE", "W"]].copy()

...: nba\_players\_df = nba\_players\_df.merge( pie\_df\_subset, how="inner", on="PLAYER")

...:

In [116]: salary\_df.rename(columns={'NAME': 'PLAYER'}, inplace=True)

...: salary\_df["SALARY\_MILLIONS"] =\ round(salary\_df["SALARY"]/1000000, 2)

...: salary\_df.drop(["POSITION","TEAM", "SALARY"],

inplace=True, axis=1)

...:

In [117]: salary\_df.head() Out[117]:

PLAYER SALARY\_MILLIONS

1. LeBron James 30.96
2. Mike Conley 26.54
3. Al Horford 26.54
4. Dirk Nowitzki 25.00
5. Carmelo Anthony 24.56

The salary information is missing for 111 NBA players, so these will be players we will drop as well when we do an analysis.

**Click here to view code image**

In [118]: diff = list(set(

nba\_players\_df["PLAYER"].values.tolist()) – set(salary\_df["PLAYER"].values.tolist()))

In [119]: len(diff)

Out[119]: 111

In [120]: nba\_players\_with\_salary\_df =\ nba\_players\_df.merge(salary\_df);

What’s left is a Pandas DataFrame with 38 columns.

**Click here to view code image**

In [121]: nba\_players\_with\_salary\_df.columns Out[121]:

Index(['Rk', 'PLAYER', 'POSITION', 'AGE', 'MP',

'FG', 'FGA', 'FG%', '3P',

'3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%', 'ORB',

'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'POINTS', 'TEAM', 'GP',

'MPG', 'ORPM', 'DRPM', 'RPM', 'WINS\_RPM', 'PIE', 'PACE', 'W',

'SALARY\_MILLIONS'],

dtype='object')

In [122]: len(nba\_players\_with\_salary\_df.columns)

Out[122]: 38

Next, the DataFrame can be merged with Wikipedia data. The data is collapsed into a median field so it can be represented as one row in a column.

**Click here to view code image**

In [123]: wiki\_df = pd.read\_csv( "../data/nba\_2017\_player\_wikipedia.csv")

In [124]: wiki\_df.rename(columns=\

{'names': 'PLAYER', "pageviews": "PAGEVIEWS"}, inplace=True) In [125]: median\_wiki\_df = wiki\_df.groupby("PLAYER").median()

In [126]: median\_wiki\_df\_small = median\_wiki\_df[["PAGEVIEWS"]] In [127]: median\_wiki\_df\_small.reset\_index(

level=0, inplace=True);median\_wiki\_df\_sm.head() Out[127]:

|  |  |  |
| --- | --- | --- |
|  | PLAYER | PAGEVIEWS |
| 0 | A.J. Hammons | 1.0 |
| 1 | Aaron Brooks | 10.0 |
| 2 | Aaron Gordon | 666.0 |

1. Aaron Harrison 487.0
2. Adreian Payne 166.0

In [128]: nba\_players\_with\_salary\_wiki\_df =\ nba\_players\_with\_salary\_df.merge(median\_wiki\_df\_small)

The final columns to add are values from the Twitter data.

**Click here to view code image**

In [129]: twitter\_df = pd.read\_csv( "../data/nba\_2017\_twitter\_players.csv")

In [130]: nba\_players\_with\_salary\_wiki\_twitter\_df=\ nba\_players\_with\_salary\_wiki\_df.merge(twitter\_df)

There are total of 41 attributes to work with now.

**Click here to view code image**

In [132]: len(nba\_players\_with\_salary\_wiki\_twitter\_df.columns)

Out[132]: 41

A logical next step in exploring the data is to create a correlation heatmap.

**Click here to view code image**

In [133]: plt.subplots(figsize=(20,15))

...: ax = plt.axes()

...: ax.set\_title("NBA Player Correlation Heatmap")

...: corr = nba\_players\_with\_salary\_wiki\_twitter\_df.corr()

...: sns.heatmap(corr,

...: xticklabels=corr.columns.values,

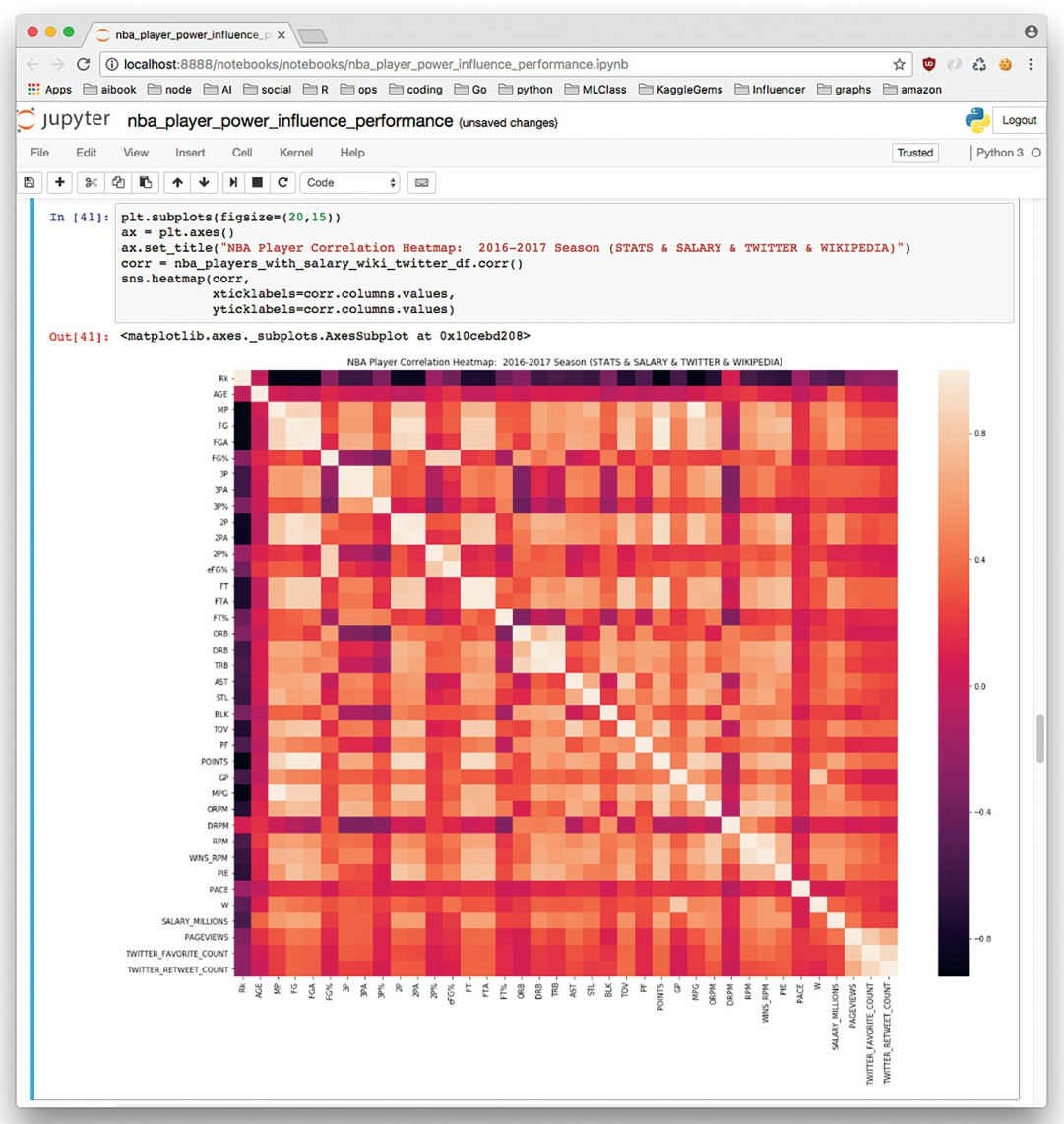
...: yticklabels=corr.columns.values)

...:

Out[133]: <matplotlib.axes.\_subplots.AxesSubplot at 0x111e665c0>

<matplotlib.figure.Figure at 0x111e66780>

[Figure 6.11](#_bookmark16) shows some fascinating correlations. Twitter engagement and Wikipedia pageviews are highly correlated. Wins attributed to player, or WINS\_RPM, is also correlated with Twitter and Wikipedia. Salary and points are highly correlated as well.



**Figure 6.11** NBA Players Correlation Heatmap: 2016–2017

# Unsupervised Machine Learning on NBA Players

With a diverse data set and many useful attributes, performing unsupervised ML on NBA players could prove to be very informative. A first step is scale the data and select the attributes against which to cluster (dropping rows with any missing values).

**Click here to view code image**

In [135]: numerical\_df =\ nba\_players\_with\_salary\_wiki\_twitter\_df.loc[:,\

["AGE", "TRB", "AST", "STL", "TOV", "BLK", "PF", "POINTS",\ "MPG", "WINS\_RPM", "W", "SALARY\_MILLIONS", "PAGEVIEWS", \

"TWITTER\_FAVORITE\_COUNT"]].dropna()

In [142]: from sklearn.preprocessing import MinMaxScaler

...: scaler = MinMaxScaler()

...: print(scaler.fit(numerical\_df))

...: print(scaler.transform(numerical\_df))

...:

MinMaxScaler(copy=True, feature\_range=(0, 1))

[[ 4.28571429e-01 8.35937500e-01 9.27927928e-01 ..., 2.43447079e-01 1.73521746e-01]

[ 3.80952381e-01 6.32812500e-01 1.00000000e+00 ..., 1.86527023e-01 7.89216485e-02]

[ 1.90476190e-01 9.21875000e-01 1.80180180e-01 ..., 4.58206449e-03 2.99723082e-02]

...,

[ 9.52380952e-02 8.59375000e-02 2.70270270e-02 ..., 1.52830350e-02 8.95911386e-04]

[ 2.85714286e-01 8.59375000e-02 3.60360360e-02 ..., 1.19532117e-03 1.38459032e-03]

[ 1.42857143e-01 1.09375000e-01 1.80180180e-02 ..., 7.25730711e-03 0.00000000e+00]]

Next, let’s cluster again and write out a CSV file to do faceted plotting in R.

**Click here to view code image**

In [149]: from sklearn.cluster import KMeans

...: k\_means = KMeans(n\_clusters=5)

...: kmeans = k\_means.fit(scaler.transform(numerical\_df))

...: nba\_players\_with\_salary\_wiki\_twitter\_df['cluster'] = kmeans.labels\_

...:

In [150]: nba\_players\_with\_salary\_wiki\_twitter\_df.to\_csv( "../data/nba\_2017\_players\_social\_with\_clusters.csv")

## Faceting Cluster Plotting in R on NBA Players

First, import the CSV file and use the ggplot2 library.

**Click here to view code image**

* player\_cluster <- read\_csv(

+ "nba\_2017\_players\_social\_with\_clusters.csv",

+ col\_types = cols(X1 = col\_skip()))

* library("ggplot2")

Next, give all four clusters meaningful names.

**Click here to view code image**

* #Name Clusters
* player\_cluster$cluster\_name[player\_cluster$

+ cluster == 0] <- "Low Pay/Low"

* player\_cluster$cluster\_name[player\_cluster$

+ cluster == 1] <- "High Pay/Above Average Performance"

* player\_cluster$cluster\_name[player\_cluster$

+ cluster == 2] <- "Low Pay/Average Performance"

* player\_cluster$cluster\_name[player\_cluster$

+ cluster == 3] <- "High Pay/High Performance"

* player\_cluster$cluster\_name[player\_cluster$

+ cluster == 4] <- "Medium Pay/Above Average Performance"

Create facets with the cluster names.

**Click here to view code image**

* #Create faceted plot
* p <- ggplot(data = player\_cluster) +

+ geom\_point(mapping = aes(x = WINS\_RPM,

+ y = POINTS,

+ color = SALARY\_MILLIONS,

+ size = PAGEVIEWS))+

+ facet\_wrap(~ cluster\_name) +

+ ggtitle("NBA Players Faceted") +

+ ylab("POINTS PER GAME") +

+ xlab("WINS ATTRIBUTABLE TO PLAYER (WINS\_RPM)") +

+ geom\_text(aes(x = WINS\_RPM, y = POINTS,

There is a bit of work to figure plot text in each facet, and this is accomplished by R and/or statements below. There is also the use of three colors in the salary, which allows for a much clearer view of the differences.

**Click here to view code image**

label=ifelse(

+ PAGEVIEWS>10000|TOV>5|AGE>37|WINS\_RPM>15|cluster

+ == 2 & WINS\_RPM > 3,

+

as.character(PLAYER),'')),hjust=.8, check\_overlap = FALSE)

>

* #Change legends
* p +

+ guides(color = guide\_legend(title = "Salary Millions")) +

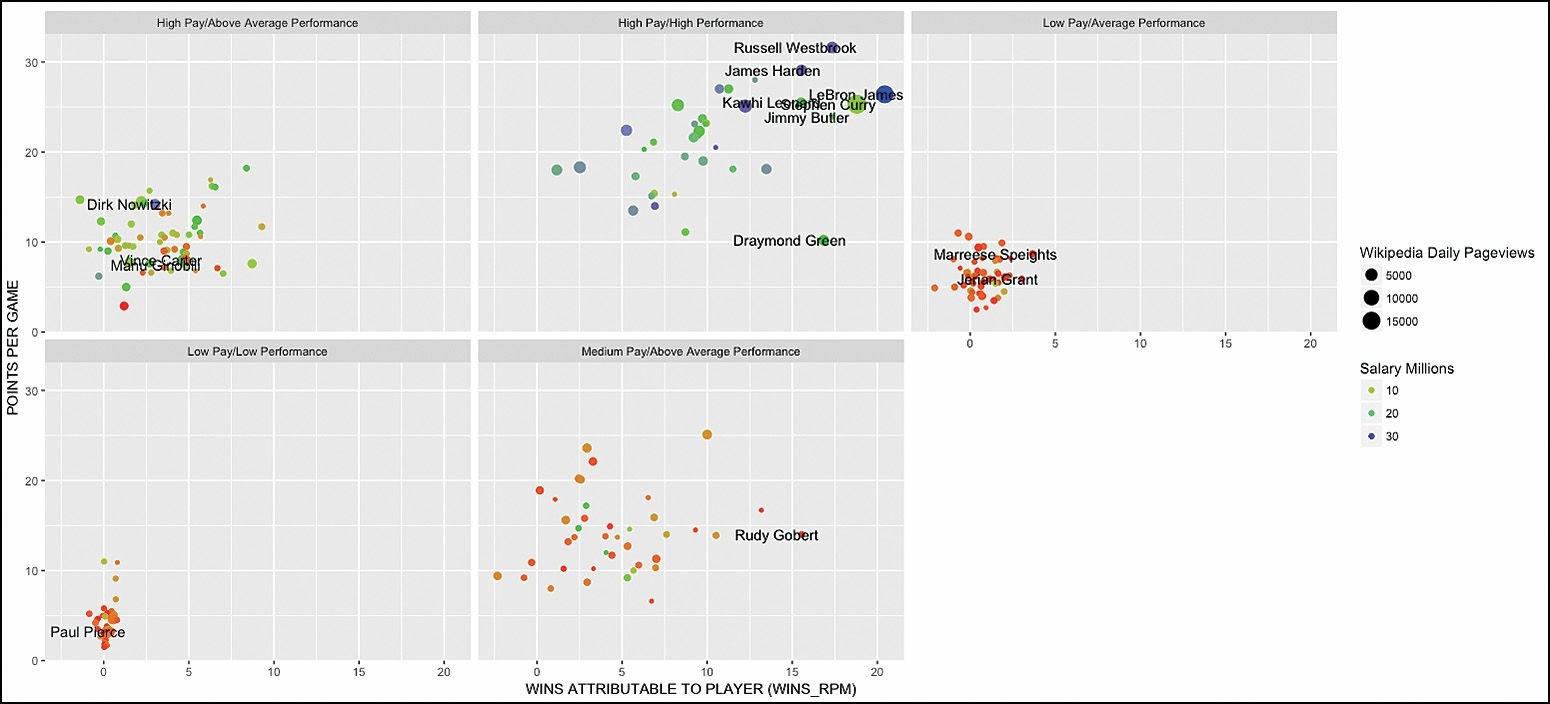
+ guides(size = guide\_legend(

+ title = "Wikipedia Daily Pageviews" ))+

+ scale\_color\_gradientn(colours = rainbow(3))

* geom\_text(aes(x = ELO, y = VALUE\_MILLIONS, label=ifelse( VALUE\_MILLIONS>1200,as.character(TEAM),'')),hjust=.35,vjust=1)

The final result is a nifty, faceted plot as shown in [Figure 6.12](#_bookmark17). The main labels that have been discovered are the differences between popularity, salary, and performance. The cluster with LeBron James and Russell Westbrook has the “best of the best,” but they also command the highest salaries.



**Figure 6.12** ggplot Faceted Plot NBA Players: 2016–2017 with kNN

## Putting it Al Together: Teams, Players, Power, and Endorsements

With all the data collected, there are some interesting new plots to test out. By combining the endorsement, team, and player data, it is possible to make a couple of fascinating plots. First, the endorsement data can be shown in a correlation heatmap in [Figure 6.13](#_bookmark18). You can see the “copper” color adds an interesting twist to this plot.

**Click here to view code image**

In [150]: nba\_players\_with\_salary\_wiki\_twitter\_df.to\_csv( "../data/nba\_2017\_players\_social\_with\_clusters.csv")

In [151]: endorsements = pd.read\_csv( "../data/nba\_2017\_endorsement\_full\_stats.csv")

In [152]: plt.subplots(figsize=(20,15))

...: ax = plt.axes()

...: ax.set\_title("NBA Player Endorsement, \ Social Power, On-Court Performance, \

Team Valuation Correlation Heatmap: 2016-2017

...: Season")

...: corr = endorsements.corr()

...: sns.heatmap(corr,

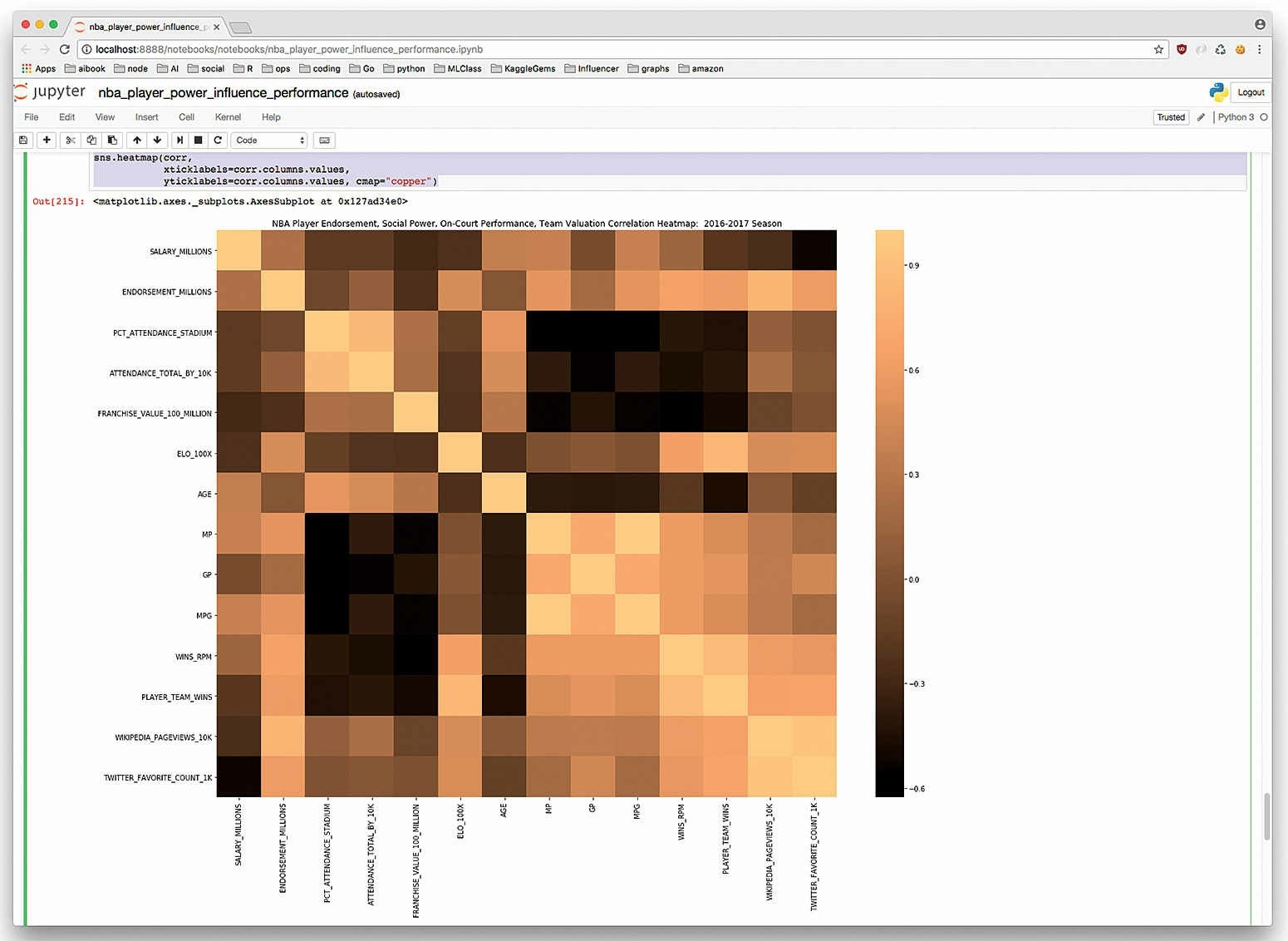
...: xticklabels=corr.columns.values,

...: yticklabels=corr.columns.values, cmap="copper")

...:

Out[152]: <matplotlib.axes.\_subplots.AxesSubplot at 0x1124d90b8>

<matplotlib.figure.Figure at 0x1124d9908>



**Figure 6.13** Endorsements Correlation Heatmap

Next, in an accent plot, the totality of the work is showcased in [Figure 6.14](#_bookmark19). The code for that is

**Click here to view code image**

In [153]: from matplotlib.colors import LogNorm

...: plt.subplots(figsize=(20,15))

...: pd.set\_option('display.float\_format', lambda x: '%.3f' % x)

...: norm = LogNorm()

...: ax = plt.axes()

...: grid = endorsements.select\_dtypes([np.number])

...: ax.set\_title("NBA Player Endorsement,\ Social Power, On-Court Performance,\

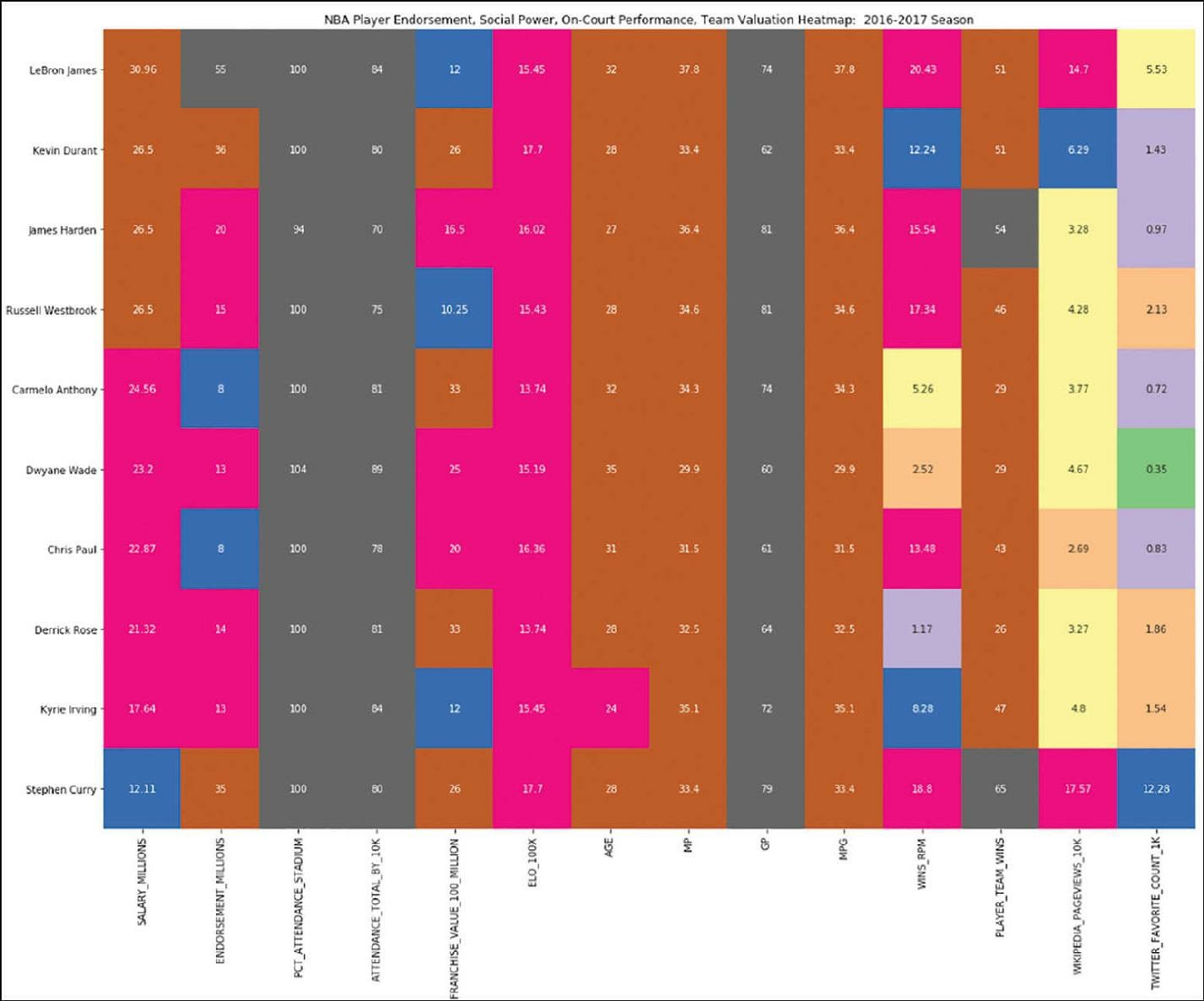
Team Valuation Heatmap: 2016-2017 Season")

...: sns.heatmap(grid,annot=True, yticklabels=endorsements["PLAYER"],fmt='g', cmap="Accent", cbar=False, norm=norm)

...:

Out[153]: <matplotlib.axes.\_subplots.AxesSubplot at 0x114902cc0>

<matplotlib.figure.Figure at 0x114902048>



**Figure 6.14** Endorsements for Players, Accent Plot

Note that a huge part of making the accent plot readable is converting the colors to LogNorm. This allows relative changes to be the catalyst for boundaries between cells.

# Further Pragmatic Steps and Learnings

One of the key reasons for this book to exist is to show how to create complete working solutions deployable to production. One way to get this solution out of a notebook would be to explore some of the solutions in other chapters that go over techniques to get projects shipped into production, for example, creating an NBA Team Valuation prediction API, or an API that showed the social power of NBA superstars. A Y combinator (YC) pitch deck might be just a few more lines of code away.

[In addition to that, a Kaggle notebook can be forked (https://www.kaggle.com/noahgift/social-power- nba), and that could be starting point for even more exploration. Finally, a video and slides on this](https://www.kaggle.com/noahgift/social-power-nba) topic can be found on the Strata Data Conference 2018 San Jose schedule:

<https://conferences.oreilly.com/strata/strata-ca/public/schedule/detail/63606>.

# Summary

This chapter looked at a real-world ML problem, starting with questions and then moving into techniques on how to collect data from all over the internet. Many of the smaller data sets were cut and pasted from web sites that may or may not have been friendly to their collection. The larger data sources Wikipedia and Twitter required a different approach—a more software engineering–centric approach.

Next, the data was explored in both a statistical fashion and using unsupervised ML and data visualization. In the final section, several solutions were created using cloud providers, including a scalable API, a serverless application, and a data visualization framework (Shiny).