

# Project2\_johnfernow\_hayleyleblanc

## 1 Project 2: Relational Databases

Hayley LeBlanc and John Fernow

Sean Lahman's baseball database is one of the most detailed and comprehensive baseball databases available online. It contains data about players, teams, statistics, and more from 1871 up to the present. It contains 27 tables organized into a relational database that can be accessed using SQL. This relational setup makes data easy to access and, more importantly, makes it easy to compare data across different tables.

In this iPython notebook, we use Sean Lahman's database to create and answer questions about baseball. We use SQLAlchemy to query the database and collect relevant information, then use Tableau to create visualizations that make the answers to our questions clear. The notebook includes SQL and SQLAlchemy statements, schemas of the parts of the database that we use, and graphs of the data with explanations.

In order to access the database from this ipynb, we must import the necessary functions from the `sqlalchemy` module so that we can create SQL queries in Python. Throughout this notebook, raw SQL queries are shown before their matching `sqlalchemy` counterparts. However, only the `sqlalchemy` statements are used to query the database. We also use `pandas` to read the queries, executing them and creating `DataFrames` that can then be written to `csv` files and used by Tableau. The `import` statements, as well as the creation of the connection to the server, are in the following cell.

```
[1]: from sqlalchemy import create_engine, Table, MetaData, select, func, distinct, \
    ↪ and_, or_
import pandas as pd
protocol = "mysql+mysqlconnector"
userid = "studen_j1"
userpass = "studen_j1"
mysqlhost = "hadoop2.mathsci.denison.edu"
database = "lahman2016"
connectionstring = "{}://{}:{}_@{}/{}".format(protocol, userid, userpass, \
    ↪mysqlhost, database)

engine = create_engine(connectionstring)
metadata = MetaData()
```

Next, we create the `sqlalchemy` representations of the tables in the database that we want to use for the first couple questions. This is done early in the notebook so that we can reuse important

tables (like `master`) multiple times without recreating the tables.

```
[2]: # create the necessary tables for the SQL Alchemy statements to use later
master = Table("Master", metadata, autoload=True, autoload_with=engine)
batting = Table("Batting", metadata, autoload=True, autoload_with=engine)
salaries = Table("Salaries", metadata, autoload=True, autoload_with=engine)
teams = Table("Teams", metadata, autoload=True, autoload_with=engine)
```

The first thing that we want to see is whether a player's height and weight affect their batting average. Batting average is one way to measure a player's skill and success, so we chose it as the metric with which to compare players of different sizes for this question. However, it is important to note that the weights, heights, and batting averages here are averaged over each player's entire career, so if a player lost weight or got much better over the course of his career, the change would not be reflected well in the graph. Also, players who were only at bat once in their career and hit the ball have a batting average of 0, which may make them appear to be more skilled players than they actually are. This data also excludes data points where a player was never at bat, because it is impossible to calculate a batting average for that point.

The following cell contains the raw SQL statement that returns the information necessary to answer this question, as well as the `sqlalchemy` version of the statement. The information returned by this statement is written into a `.csv` called `question1.csv`. The first 10 rows of the `pandas DataFrame` created using data from the query are also printed below.

```
[3]: # Does a player's height and weight affect their career batting average?

# RAW SQL
# SELECT Master.playerID, Master.nameFirst, Master.nameLast, Master.weight, \
↳ Master.height,
#     AVG(Batting.H/Batting.AB) AS BattingAverage
# FROM (Master
#     INNER JOIN Batting
#     ON Master.playerID = Batting.playerID)
# WHERE Master.weight <> 0 AND
#     Master.weight IS NOT NULL AND
#     Master.height <> 0 AND
#     Master.height IS NOT NULL AND
#     Batting.AB <> 0
#     Batting.H IS NOT NULL AND
#     Batting.AB IS NOT NULL
# GROUP BY Master.playerID
# ORDER BY Master.nameLast;

# SQL Alchemy
with engine.connect() as connection:
    stmt = select([master.c.playerID, master.c.weight, master.c.height, \
                    func.avg(batting.c.H/batting.c.AB).label("BattingAverage")])
    stmt = stmt.select_from(master.join(batting, master.c.playerID==batting.c.
↳ playerID))
```

```

stmt = stmt.where(and_(master.c.weight != 0, master.c.weight != None))
stmt = stmt.where(and_(master.c.height != 0, master.c.height != None))
stmt = stmt.where(batting.c.AB != 0)
stmt = stmt.where(and_(batting.c.H != None, batting.c.AB != None))
stmt = stmt.group_by(master.c.playerID)
stmt = stmt.order_by(master.c.playerID)
df = pd.read_sql_query(stmt, engine)
# set the DataFrame's index to get rid of the automatically generated index_
→column
df = df.set_index(["playerID"])

df.to_csv("question1.csv")
df.head(10)

```

```

[3]:
      weight  height  BattingAverage
playerID
aardsda01    215     75         0.000000
aaronha01    180     72         0.301065
aaronto01    190     75         0.220643
aasedo01     190     75         0.000000
abadan01     184     73         0.039200
abadfe01     220     73         0.047633
abadijo01    192     72         0.236100
abbated01    170     71         0.227880
abbeybe01    175     71         0.211000
abbeych01    169     68         0.274120

```

The following cell is a schema drawing of the tables and attributes used to access this data.

The following graphs were created using data from this query.

Both of these scatter plots show fairly similar shapes. Most players are between 66 and 80 inches (5.5-6.67 feet) tall, and most players weigh between about 130 and 240 pounds, so most of the data is between those two ranges. We can also see that most players have a batting average below about 0.40, because most of the data is also consolidated behind that point. There are also a fair number of players with a batting average of 0 (i.e., they never hit a pitch over the course of their career), and some players who have a batting average of 1. However, as was stated before, most of those players were only at bat a couple times in over the course of their career, so their batting averages don't give us as much information. We will ignore these players with batting averages of 1 for the rest of the analysis of these two graphs.

According to these graphs, there seems to be a bit of a bottleneck in weight and height when we look at batting averages above 0.40. Up to that point, players are pretty varied in size, but past 0.40, most players tend to be between 68 and 76 inches (5.67-6.3 feet) tall and weigh between 160 and 220 pounds. So, one could possibly conclude that players who are of a more median size, rather than being on the larger or smaller ends, are more likely to have a better batting average. However, this finding could also be a result of the fact that there are more people in general who are in that range of height and weight, so there have been more baseball players in that range, so it would make sense that there are more players with good batting averages in that range. We can

conclude, however, that being especially large or especially small does not lend one any particular advantage when it comes to batting average.

Interestingly, there is a clear line at 0.50 in both graphs that seems to show that 0.50 is a fairly common batting average despite the fact that the points on the plot begin to thin out at 0.40. I would guess that this line is caused by a fairly significant number of players only batting several times throughout their career (2-4 times) and getting hits half of that. This brings up another potential issue with these graphs, which is that players who batted more than once but only a couple times over the course of their career may have inflated batting averages.

The next thing we wanted to see was where the highest-paid baseball players come from. We use two SQL statements to find this. The first determines the average salaries for players from each state that appears in the database, and the second determines the average salaries for players from each country that appears in the database. The raw SQL statement and the `sqlalchemy` version are both in the following cells. The information from the queries is written into `question2_1.csv` and `question2_2.csv`. The first 10 rows of the `pandas DataFrame` created using data from the query are also printed below.

```
[4]: # Which states do the highest-paid baseball players come from?

# RAW SQL
# SELECT Master.playerID, AVG(Salaries.salary) AS AverageSalary, Master.
    ↳ birthState
#     FROM (Master
#           INNER JOIN Salaries
#           ON Master.playerID = Salaries.playerID)
#     GROUP BY Master.playerID

# SQL Alchemy
with engine.connect() as connection:
    stmt1 = select([master.c.playerID, func.avg(salaries.c.salary).
    ↳ label("AvgSalary"), master.c.birthState])
    stmt1 = stmt1.select_from(master.join(salaries, master.c.playerID==salaries.
    ↳ c.playerID))
    stmt1 = stmt1.group_by(master.c.playerID)

    df1 = pd.read_sql_query(stmt1, engine)
    # set the DataFrame's index to get rid of the automatically generated index_
    ↳ column
    df1 = df1.set_index(["playerID"])

df1.to_csv("question2_1.csv")
df1.head(10)
```

```
[4]:           AvgSalary birthState
playerID
aardsda01  1.322821e+06         CO
```

aasedo01	5.750000e+05	CA
abadan01	3.270000e+05	FL
abadfe01	7.532800e+05	La Romana
abbotje01	2.462500e+05	GA
abbotji01	1.440056e+06	MI
abbotku01	4.707778e+05	OH
abbotky01	1.295000e+05	MA
abbotpa01	9.244286e+05	CA
abercree01	3.270000e+05	GA

```
[5]: # Which countries do the highest-paid baseball players come from?

# RAW SQL
# SELECT Master.playerID, AVG(Salaries.salary) AS AverageSalary, Master.
    ↳ birthCountry
# FROM (Master
# INNER JOIN Salaries
# ON Master.playerID = Salaries.playerID)
# GROUP BY Master.playerID

# SQL Alchemy
with engine.connect() as connection:
    stmt2 = select([master.c.playerID, func.avg(salaries.c.salary).
    ↳ label("AverageSalary"), master.c.birthCountry])
    stmt2 = stmt2.select_from(master.join(salaries, master.c.playerID==salaries.
    ↳ c.playerID))
    stmt2 = stmt2.group_by(master.c.playerID)

    df2 = pd.read_sql_query(stmt2, engine)
    # set the DataFrame's index to get rid of the automatically generated index
    ↳ column
    df2 = df2.set_index(["playerID"])

df2.to_csv("question2_2.csv")
df2.head(10)
```

```
[5]:
```

	AverageSalary	birthCountry
playerID		
aardsda01	1.322821e+06	USA
aasedo01	5.750000e+05	USA
abadan01	3.270000e+05	USA
abadfe01	7.532800e+05	D.R.
abbotje01	2.462500e+05	USA
abbotji01	1.440056e+06	USA
abbotku01	4.707778e+05	USA
abbotky01	1.295000e+05	USA
abbotpa01	9.244286e+05	USA

abercree01    3.270000e+05            USA

The following cell is a schema drawing of the tables and attributes used to access this data.

The following graphs were created using data from these queries.

This graph shows us where the highest-paid players are from worldwide. Each player's average salary over their career was taken and used to compute the average salary per country. Greener countries have higher average salaries, while redder countries have lower average salaries. According to this map, the highest-paid players come from Japan, Vietnam, and Cuba, while the lowest-paid players come from Spain, Belgium, and Afghanistan. Baseball is extremely popular in Japan and Cuba, which explains why many high-paid players come from those two countries. There is only one player in Lahman's database who was born in Vietnam - Danny Graves, who actually lived in America for most of his life. This brings up several potential issues with this visualization. First, it does not account for WHERE players grew up and learned baseball - it simply tells us where they were born. This means that some conclusions that could potentially be drawn from this map may be incorrect. For example, one could conclude that baseball is popular in Vietnam and that the country produces many players who are successful in the MLB, when in fact the opposite is true. Second, since the salaries in Lahman's database are most likely not adjusted for inflation, countries in which baseball became popular more recently may appear to have higher-paid players in the MLB. For example, Vietnam's average player salary is so high because Danny Graves played in the 1990's and 2000's, meaning that his average salary is much higher than players from the 1800s or early 1900s. This means that countries that did produce good players for American baseball earlier in its history may have lower average salaries than those that got into it more recently.

This graph shows us where the highest-paid players are from in the United States. It excludes players who were born outside of the United States. The greener a state is, the higher players who were born there are paid; the redder a state is, the lower players who were born there are paid. Some of the issues with the previous graph are not as impactful here, because it is much more likely that all states have had some involvement with baseball for longer than all countries. However, it is still true that many of the salaries in the database are not adjusted for inflation, so that could impact the accuracy of this visualization.

From this map, we can see that the highest-paid players were born in North Dakota, Arkansas, and Rhode Island, while the lowest-paid players were born in Montana, Utah, and Vermont. Interestingly, none of these states currently have their own MLB teams, so these players must have gone away from home for their professional baseball careers.

For the next question, we wanted to see if teams in different leagues have different overall batting averages. Lahman's database contains teams from the two current leagues, the American League and the National League, as well as several other now-defunct leagues (the American Association, the Union Association, the Player's League, and the Federal League). I have chosen to only compare teams from only the American League and the National League. The main difference between these two leagues is that the American League allows each team to have a designated hitter who bats instead of the pitcher (since pitchers are often poor batters). This rule was instituted in 1973, so this query only returns data from 1973 to the present so that we can see the effects of this rule. The results are outputted to the file `question3.csv`. The first 10 rows of the `pandas DataFrame` created using data from the query are also printed below.

```
[6]: # Do teams in the American League have higher average batting averages than
      ↪ teams in the National League?

# RAW SQL
# SELECT Teams.teamID, Teams.name, Teams.lgID, AVG(Batting.H/Batting.AB) AS
      ↪ AvgBattingAverage
# FROM (Teams
# INNER JOIN Batting
# ON Teams.teamID = Batting.teamID)
# WHERE Batting.AB <> 0 AND
# Batting.H IS NOT NULL AND
# Batting.AB IS NOT NULL AND
# (Teams.lgID = 'AL' OR Teams.lgID = 'NL')
# GROUP BY Teams.teamID
# ORDER BY Teams.lgID;

# SQLAlchemy
with engine.connect() as connection:
    stmt = select([teams.c.teamID, teams.c.name, teams.c.lgID, \
                  func.avg(batting.c.H/batting.c.AB).
      ↪ label("AvgBattingAverage")])
    stmt = stmt.select_from(teams.join(batting, teams.c.teamID==batting.c.
      ↪ teamID))
    stmt = stmt.where(and_(batting.c.AB != 0, batting.c.H != None))
    stmt = stmt.where(batting.c.AB != None)
    stmt = stmt.where(or_(teams.c.lgID=="AL", teams.c.lgID=="NL"))
    stmt = stmt.where(teams.c.yearID >= 1973)
    stmt = stmt.group_by(teams.c.teamID)
    stmt = stmt.order_by(teams.c.lgID)

    df = pd.read_sql_query(stmt, engine)
    # set the DataFrame's index to get rid of the automatically generated index
      ↪ column
    df = df.set_index(["teamID"])

df.to_csv("question3.csv")
df.head(10)
```

```
[6]:
```

	name	lgID	AvgBattingAverage
teamID			
CLE	Cleveland Indians	AL	0.219113
DET	Detroit Tigers	AL	0.213999
BOS	Boston Red Sox	AL	0.216389
CHA	Chicago White Sox	AL	0.212699
NYA	New York Yankees	AL	0.216017
BAL	Baltimore Orioles	AL	0.210021
MIN	Minnesota Twins	AL	0.219943

CAL	California Angels	AL	0.216710
OAK	Oakland Athletics	AL	0.206951
KCA	Kansas City Royals	AL	0.222553

The following cell is a schema drawing of the tables and attributes used to access this data.

The following graphs were created using data from this query.

First, one may note that the Milwaukee Brewers are present in both tables. The Brewers team was founded in 1969 and was originally a member of the American League, but moved to the National League in 1998. So, the batting average for the Brewers in the American League graph accounts for their playing in 1973-1998, and their batting average in the National League graph accounts for their playing in 1998-2016.

We can see in the above graphs that in general, American League teams tend to have higher overall batting averages since 1973 than National League teams do. The American League teams' batting averages sit between 0.195 and 0.235, while the National League teams are between 0.180 and 0.210. Although we don't know what other factors may have affected these teams between 1973 and 2016, it seems as though the designated hitter rule has a pretty big impact on teams' overall batting averages. Since the American League allows for designated hitters, teams can train one player to be an excellent hitter while allowing their pitcher to practice his throwing, which gives them a higher batting average. In contrast, the National League requires pitchers to bat, but since they most likely focus more on becoming a better pitcher than a better hitter, they could bring their teams' batting averages way down.

Since the two different leagues have different rules, what do teams do when they play inter-league games, like in the World Series? The rules have changed several times since the designated hitter rule was instated, but since the 80's the policy has been that the rule is used whenever the game is played in an American League stadium, and is not used when the game is played in a National League stadium.

One important question to ask when considering the economics of baseball is whether a higher ranking results in higher attendance (and thus, usually, higher profits). Thus the following function was written to determine that answer. To keep it relevant to today, it was limited to 2016 (the most recent year available).

[7]: *# Do teams with higher ranks get significantly higher attendance? (in 2016)*

```
Teams = Table('Teams',metadata,autoload=True, autoload_with=engine)
rawSQL = """
SELECT DISTINCT teamID, attendance, rank
FROM Teams
WHERE attendance > 0 AND yearID = 2016
GROUP BY teamID;
"""

with engine.connect() as connection:
    stmt = select([Teams.c.teamID,
                  Teams.c.attendance,
                  Teams.c.Rank])
```



```

stmt = stmt.distinct()

stmt = stmt.where(Teams.c.attendance > 0)

stmt = stmt.where(Teams.c.yearID == 2016)

results = connection.execute(stmt).fetchall()
df = pd.DataFrame(results)
df.columns = results[0].keys()

# exported to CSV to use in Tableau
df.to_csv('attendance.csv', index=False)


df.head(10)

```

```

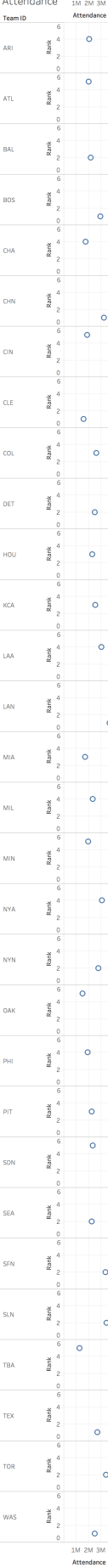
[7]:  teamID attendance Rank
0     ARI      2036216     4
1     ATL      2020914     5
2     BAL      2172344     2
3     BOS      2955434     1
4     CHA      1746293     4
5     CHN      3232420     1
6     CIN      1894085     5
7     CLE      1591667     1
8     COL      2602524     3
9     DET      2493859     2

```

Teams		
	teamID	varchar
	attendance	varchar
	Rank	integer

As you can see from the chart below, a higher rank does usually result in higher attendance. However, there are a fair number of outliers, with some teams that are ranked quite well getting poor attendance while other teams getting high attendance

Impact of Rank on Attendance



despite a poor ranking.

Sum of Attendance vs. sum of Rank broken down by Team ID.

Another important question to ask was whether teams that paid their best players better had more people in the Hall of Fame. Note: the following dataframe contains repeats, as player salary varies year to year, and some players played for multiple teams. This, however, does not change the true measurement here, which is to see if teams who pay their players better have more members in the hall of fame, as everything is averaged out for the team.

```
[8]: # If teams pay their best players more, do they have more players in the Hall
      ↳ of Fame?

HallOfFame = Table('HallOfFame',metadata,autoload=True, autoload_with=engine)
Salaries = Table('Salaries',metadata,autoload=True, autoload_with=engine)

rawSQL = """
SELECT DISTINCT HallOfFame.playerID, Salaries.salary, Salaries.teamID
FROM HallOfFame
INNER JOIN Salaries ON HallOfFame.playerID = Salaries.playerID;
"""

with engine.connect() as connection:
    stmt = select([HallOfFame.c.playerID,
                    Salaries.c.salary,
                    Salaries.c.teamID])

    stmt = stmt.select_from(
        HallOfFame.join(Salaries, HallOfFame.c.playerID == Salaries.c.playerID)
    )

    stmt = stmt.distinct()

    results = connection.execute(stmt).fetchall()
    df = pd.DataFrame(results)
    df.columns = results[0].keys()

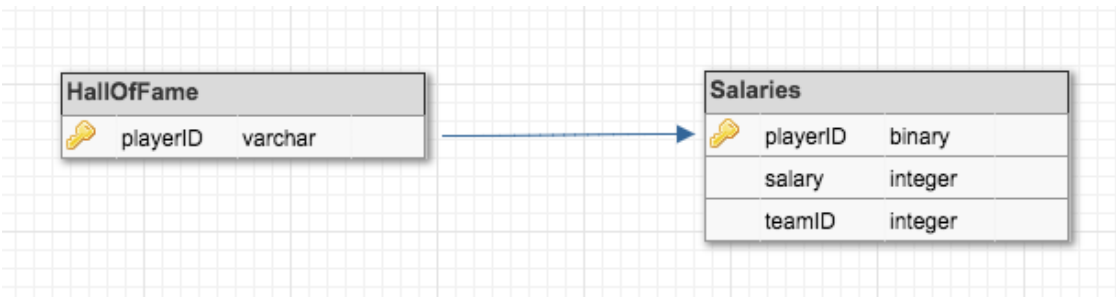
# exported to CSV to use in Tableau
df.to_csv('hall_of_fame.csv',index=False)

df.head(10)
```

```
[8]:
```

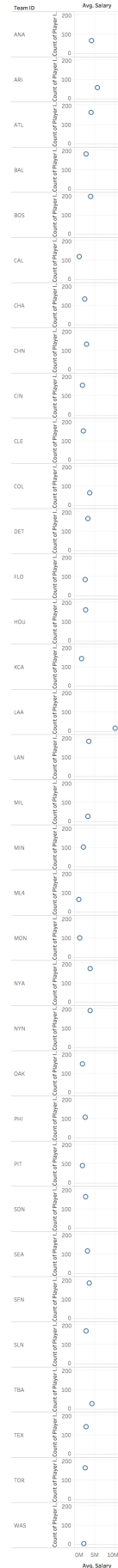
	playerID	salary	teamID
0	remyje01	483333	BOS
1	carewro01	875000	CAL
2	goltzda01	500000	LAN
3	ziskri01	272500	SEA
4	chambch01	800000	ATL
5	hornebo01	1500000	ATL
6	suttebr01	1354167	ATL
7	dauerri01	480000	BAL

8	mcgresc01	547143	BAL
9	kisonbr01	370000	BOS



As the chart below shows, there is very poor correlation between higher salary and a greater number of players in the Hall of Fame. Some teams have spent fairly little and still gotten a significant number of players in the Hall of Fame, while others have spent a considerable amount of money and still gotten few players in the Hall of Fame. Note, inflation overall should not be a concern since this is the average salary of Hall of Fame players for each team, thus all teams are effected by inflation. However, the few teams that have very few people in the Hall of Fame are not great

Hall of Fame OM SM 10M



Baseball managers can easily be divided into two categories: player managers, and non-player managers. From this, it is easy to wonder if player managers are better, and if so, how much better? Note, the following algorithm produces a dataframe with repeats. This is necessary, as the data cannot be limited to the past year, as there has not been a single player-manager in 32 years. As such, to get a good sample of player managers, it was important to include all years recorded. As such there will be repeats, but repeats will exist for both player managers and non-player managers, so it will not skew the data (the repeats are for when a manager is a manager for more than one year, but since their rank changes year to year, it is still valuable data, and by including these repeated entries, it will account for their average over their career, not just a single year.)

```
[9]: # Are player managers better than non-player managers?

Managers = Table('Managers',metadata,autoload=True, autoload_with=engine)

rawSQL = """
SELECT DISTINCT playerID, plyrMgr, Rank
FROM Managers;
"""

with engine.connect() as connection:

    stmt = select([Managers.c.playerID,
                    Managers.c.plyrMgr,
                    Managers.c.rank])

    stmt = stmt.distinct()


    results = connection.execute(stmt).fetchall()
    df = pd.DataFrame(results)
    df.columns = results[0].keys()

# exported to CSV to use in Tableau
df.to_csv('managers.csv',index=False)

df.head(10)
```

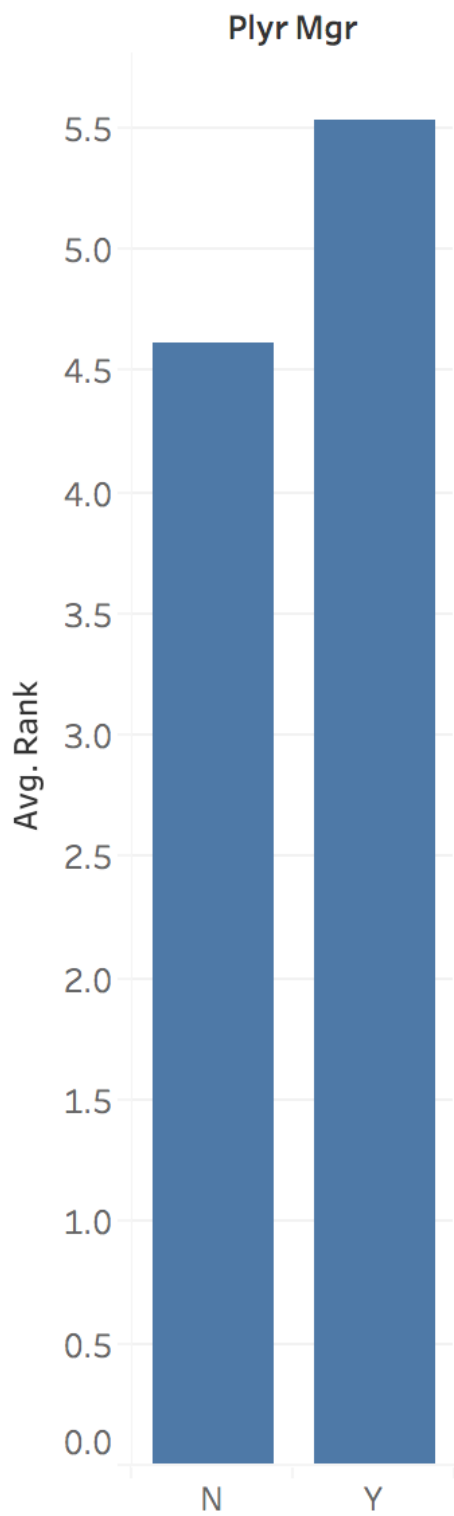
```
[9]:
```

	playerID	plyrMgr	rank
0	wrighha01	Y	3
1	woodji01	Y	2
2	paborch01	Y	8
3	lennobi01	Y	8
4	deaneha01	Y	8
5	fergubo01	Y	5
6	mcbri01	Y	1
7	hastisc01	Y	9
8	pikel01	Y	6
9	cravebi01	Y	6

Managers			
	playerID	varchar	
	plyrMgr	varchar	
	Rank	integer	

As the chart below shows, player managers, on average, are better than non-player managers. However, it is not as great as a divide as one may

# The Impact on Rank of Being a Player-Manager



16  
Average of Rank for each Plyr Mgr.

have expected.



[ ]: