A Comprehensive Analysis of NBA Player Value

Ben Brown, Cooper Nelson, Corey Becker, Ethan Richards, Justin Collier
Group 14

Our Data

What is interesting about these datasets? (Note that this can mean a lot of things - the data was for an interesting/timely topic, the data was interesting (probably difficult) to work with, the applications of the data are interesting, etc.)

Our datasets include the total stats of a player for every season and of their career and the players salaries for each year they played. The data for the total stats dates back to the creation of the NBA in 1947 and includes almost every player who has ever played in the NBA. We also used a salary dataset which contains player salaries since 1990. These datasets are interesting because we can combine them to find out which players produce the best statistics for their salary. Our final tables include all players from 1990 to the 2023 season. Our player stats table contains all traditional counting stats plus a few advanced stats such as points, rebounds, assists, field goal percentage. The data allows us to analyze the business side of basketball and find how valuable a player is based on their salary and how much improvement a more expensive player will provide. We can also analyze how NBA teams value certain players over time and where the NBA teams decide to spend their money. With this data we can determine how teams should spend their money and try to find undervalued attributes.

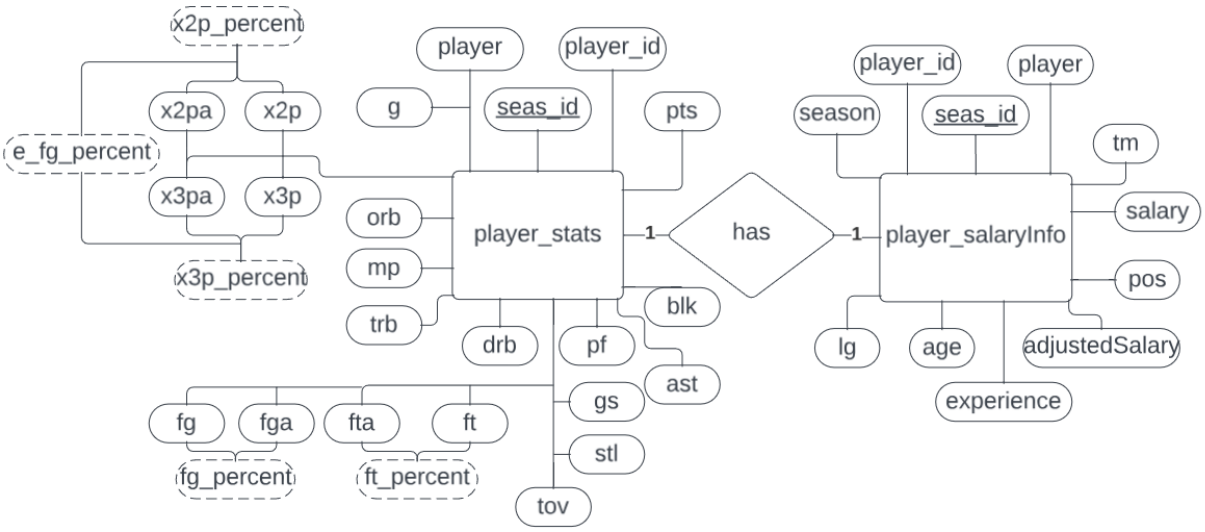
Where was the data obtained?

To get our data, we used both Kaggle and did our own web scraping. Our player statistics dataset was created by Sumitro Datta on Kaggle, and the statistics were scraped from Basketball Reference and contain information for almost all NBA players. Our NBA salary data was sourced from HoopsHype which is a USA Today sports website. We web scraped all of the information from this site and saved it as a csv file.

What license restrictions are there on the use of the data? (Note a website providing data automatically copyrights that data unless it has terms and conditions stating otherwise - be sure you have permission to use that data in your project.)

There is not a specific license on the data, but from the ToS of the player statistics website: "*This Site and all the materials available on the Site are the property of us and/or our affiliates or licensors, and are protected by copyright, trademark, and other intellectual property laws. The Site is provided solely for your personal noncommercial use.*" Therefore, our site implies that we can use this data for noncommercial use and we are making sure to adequately cite the dataset. The player salary dataset has no specific license restrictions or copyright notices on usage.

Describe the tables and significant attributes as loaded into the database, include an ER diagram of your datasets and note the interesting relationships between entities. This should be of the normalized datasets, not the original datasets, if the two differ.



We used two tables with a shared season id that is unique for each player and each season even if they change teams mid season. Each player in the player_stats table has a correlated entry in the player_salaryInfo table. The most significant attribute from the player_salaryInfo table is the adjustedSalary, but age, experience, and position were also useful. From the player_stats table, the attributes that contributed most were pts and experience.

An analysis of the at least 3NF nature of your datasets (BCNF is stronger than 3NF and can be used to meet this requirement).

All the tables are created from data representing the stats of each player for each season they played in. The statistics for each player are atomic attributes that all pertain to the same domain of information about a player's performance in a season, so the data is in first normal form. Additionally, the data relies upon a season id which is unique per season per player, so because the stats for each player are dependent on the name of the player and the season for which they are being tracked, the data is also in second normal form. Finally, there are attributes such as free throw percentage which are derived from another statistic such as free throws but none of these are dependent on another attribute. In the example of free throws, a free throw percentage can exist (and be 0) without the existence of free throws. This same relationship follows for all other derived attributes, meaning that because our data has no functional dependencies between non-key attributes it is in third normal form.

Data loading or manipulation often requires transactions to ensure an all-or-none result; what steps were taken to ensure that? How were transactions or verification of data employed?

Our data loading and manipulation did not explicitly use transactions; we used autocommit on our transactions though. We didn't especially need any transactions or verification of data employed because the cleaning happened once with one user (and no other concurrent users).If we needed to regenerate the database, that's also something we can easily do without much trouble and without incurring transaction issues.

What data cleaning operations did you perform on the datasets, and/or how did you ensure that the dataset was clean for the questions you wanted to answer?

We needed to ensure the names between the two databases were consistent because one database used non-ASCII characters in player names, while the other so we wrote a script to change these special characters to an ASCII equivalent. We also removed duplicate statistics of players who switched teams mid-season, as they had basically empty statistics for the team they left. We also cleaned salary text from the format like "$1,000,000," which broke a bit of our CSV loading originally. Finally, we adjusted our salaries for inflation and added that adjusted salary in a new column. This was necessary for answering our salary related questions.

Data is constantly changing; discuss how much your dataset might grow over the next 10 years if it were maintained, and estimate its future size based on its present size and that growth rate. This estimate should be stated in pages and take account of storage overhead (not just the data itself) as was done in class.

Potential Maximum players = 510
Player Stats = 2367488 Bytes
    13536 Rows
    2367488/8192 = 289 pages
    Pages per row = 0.02135047
Player Salary Info = 491520 Bytes
    5174 Rows
    491520/8192 = 60 pages
    Pages per row = 0.01159644
Joined Player Stats and Salary info has 5174 rows from 1991 to 2023
32 Years 5172 rows -> Average 161.625 rows a year -> 170.4014 pages now + 5.325044 pages per year
Pessimistic -> 510 rows a year -> 16.80292 pages per year

Discuss how your work meets SQL Security concerns; what permissions were used to ensure no unneeded access occurred to the data? How was SQL Injection avoided? What next steps might you use to beef up security if you continued with your project?
We stored data in a GitHub repository; this ensures that only members of the team had access to the data, and even if it was maliciously changed, we would have a history of the changes. SQL injection was avoided by not adding data to the database that came from a user. All data is pulled from published datasets so there is no opportunity for a user to inject SQL. If continuing with the project, security could be improved by verifying the site certificate in the script to scrape the site data for player salaries.

Data Manipulation and Analysis

What were the interesting queries done on your data? (at least 4) (provide English descriptions and SQL)
Question 1: What seasons are worth the most money for a player?
SQL:

```
SELECT season AS year, SUM(predsal) AS s
FROM player_reg
GROUP BY year
ORDER BY s DESC;
```

Question 2: Which players overproduced/underproduced by points per game in the 2023 season?
SQL:

```
SELECT ps.player, ps.adjustedsalary, stats.pts
FROM player_salaryinfo AS ps
JOIN player_stats AS stats ON ps.seas_id = stats.seas_id
WHERE season = 2023
GROUP BY ps.player, ps.adjustedsalary, stats.pts
ORDER BY ps.adjustedsalary DESC, stats.pts ASC;
```

Question 3: What players (who have played at least 6 years) were the most overpaid over their entire career?
SQL:

```
SELECT player, COUNT(player), SUM(residual) AS s
FROM player_reg
GROUP BY player
HAVING COUNT(player) > 5
ORDER BY s DESC;
```

Question 4: What's the highest salary per age?
SQL:

```
SELECT age, MAX(adjustedsalary)
```

FROM player_salaryinfo
GROUP BY age
ORDER BY age;

What performance improvements did you make to your schema? Show analysis of at least one query and what changes were made to improve its performance (these could be data changes, query changes, or index changes; show the EXPLAIN before improvement and after). If everything worked great from the start with performance, show the EXPLAIN and justify that it was all good as-is (for example, adding an index didn't help).

With our question/query #2, we originally had this execution with an EXPLAIN ANALYZE query:

```
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
------------------------------
 Group  (cost=641.84..645.68 rows=384 width=23) (actual time=7.249..7.470 rows=384 loops=1)
   Group Key: ps.adjustedsalary, stats.pts, ps.player
   ->  Sort  (cost=641.84..642.80 rows=384 width=23) (actual time=7.247..7.298 rows=384 loops=1)
         Sort Key: ps.adjustedsalary DESC, stats.pts, ps.player
         Sort Method: quicksort  Memory: 55kB
         ->  Hash Join  (cost=129.47..625.36 rows=384 width=23) (actual time=1.743..6.870 rows=384 loops=1)
               Hash Cond: (stats.seas_id = ps.seas_id)
               ->  Seq Scan on player_stats stats  (cost=0.00..424.36 rows=13536 width=8) (actual time=0.007..
2.027 rows=13536 loops=1)
               ->  Hash  (cost=124.67..124.67 rows=384 width=23) (actual time=1.724..1.725 rows=384 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 30kB
                     ->  Seq Scan on player_salaryinfo ps  (cost=0.00..124.67 rows=384 width=23) (actual time=
0.012..1.596 rows=384 loops=1)
                           Filter: (season = '2023'::numeric)
                           Rows Removed by Filter: 4790
 Planning Time: 0.240 ms
 Execution Time: 7.536 ms
(15 rows)
```

Then, we created an INDEX on season because we noticed that this query was filtering on if season is 2023. The result was slightly more planning time, but an improved (by about 1.5ms) execution time.

```
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
---------------------------------------
 Group  (cost=593.22..597.06 rows=384 width=23) (actual time=6.091..6.313 rows=384 loops=1)
   Group Key: ps.adjustedsalary, stats.pts, ps.player
   ->  Sort  (cost=593.22..594.18 rows=384 width=23) (actual time=6.089..6.140 rows=384 loops=1)
         Sort Key: ps.adjustedsalary DESC, stats.pts, ps.player
         Sort Method: quicksort  Memory: 55kB
         ->  Hash Join  (cost=80.86..576.74 rows=384 width=23) (actual time=0.536..5.708 rows=384 loops=1)
               Hash Cond: (stats.seas_id = ps.seas_id)
               ->  Seq Scan on player_stats stats  (cost=0.00..424.36 rows=13536 width=8) (actual time=0.006..
2.015 rows=13536 loops=1)
               ->  Hash  (cost=76.06..76.06 rows=384 width=23) (actual time=0.518..0.519 rows=384 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 30kB
                     ->  Bitmap Heap Scan on player_salaryinfo ps  (cost=11.26..76.06 rows=384 width=23) (actu
al time=0.121..0.390 rows=384 loops=1)
                           Recheck Cond: (season = '2023'::numeric)
                           Heap Blocks: exact=59
                           ->  Bitmap Index Scan on player_name  (cost=0.00..11.16 rows=384 width=0) (actual t
ime=0.100..0.100 rows=384 loops=1)
                                 Index Cond: (season = '2023'::numeric)
 Planning Time: 0.422 ms
 Execution Time: 6.391 ms
(17 rows)
```

Although this was a minor change, it definitely improved the performance of our query and would continue to work as the dataset grows.

Provide visualizations of your data or a program for accessing it; the report should contain the charts or screenshots of the running program.

We accessed some of our data in R, and this was the result. We then loaded these regressions and residuals into the database as another table to do further analysis on.

```
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -8.633e+05  3.382e+05  -2.553   0.0107 *
pts           9.500e+03  4.192e+02  22.663  < 2e-16 ***
ast           1.066e+04  1.082e+03   9.856  < 2e-16 ***
orb          -1.540e+04  2.800e+03  -5.498 4.03e-08 ***
drb           2.085e+04  1.417e+03  14.711  < 2e-16 ***
stl          -3.543e+04  4.605e+03  -7.694 1.69e-14 ***
gs            7.099e+04  5.927e+03  11.977  < 2e-16 ***
experience    6.738e+05  2.003e+04  33.643  < 2e-16 ***
mp           -6.223e+03  3.213e+02 -19.368  < 2e-16 ***
fg_percent   -1.686e+07  2.438e+06  -6.916 5.21e-12 ***
x2p_percent   5.192e+06  1.138e+06   4.561 5.21e-06 ***
e_fg_percent  1.339e+07  2.071e+06   6.463 1.12e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5342000 on 5162 degrees of freedom
Multiple R-squared:  0.4902,    Adjusted R-squared:  0.4891
F-statistic: 451.2 on 11 and 5162 DF,  p-value: < 2.2e-16
```

For our question/query #2, we got these results from the query:

player	adjustedsalary	pts
Stephen Curry	48070014.0	1648
John Wall	47345760.0	386
Russell Westbrook	47080179.0	1159
LeBron James	44474988.0	1590
Kevin Durant	44119845.0	1366
Bradley Beal	43279250.0	1160
Giannis Antetokounmpo	42492492.0	1959
Damian Lillard	42492492.0	1866
Paul George	42492492.0	1332
Kawhi Leonard	42492492.0	1239
Klay Thompson	40600080.0	1509
Kyrie Irving	38917057.0	1623
Rudy Gobert	38172414.0	939
Khris Middleton	37984276.0	497
Anthony Davis	37980720.0	1451
Kemba Walker	37281261.0	72
Luka Doncic	37096500.0	2138
Trae Young	37096500.0	1914
Zach LaVine	37096500.0	1913
Pascal Siakam	35448672.0	1720
Ben Simmons	35448672.0	291
Myles Turner	35096500.0	1113
Jrue Holiday	34319520.0	1290
Devin Booker	33833400.0	1471
Karl-Anthony Towns	33833400.0	602
Joel Embiid	33616770.0	2183
Nikola Jokic	33047803.0	1690
Jamal Murray	31650600.0	1298

In red, we highlighted players who were overpaid by the points per game they produced; John Wall and Kemba Walker were overpaid (maybe due to injury), whereas Doncic and Embiid had incredible seasons while still being paid less.

Dataset Difficulties

A section describing the technical challenges you dealt with in obtaining, manipulating, and loading the dataset(s) and/or other issues that arose in completion of the project.

Obtaining the datasets was difficult because salary datasets are not readily available nor player stats, as most of these are proprietary/commercial. Further, player salaries in sports are usually not officially released information, so our data is more of an estimate in that regard. We found one website, Basketball Reference, which had the most comprehensive player statistics, but the problem was that it didn't allow exports to csv, so we had to web scrape it.

In manipulating the dataset, we had a few main issues. Firstly, since we loaded the player salaries as a csv, the salary values that looked like "$1,000,000" were causing major problems, along with the dollar sign. We had to clean this all with a Python script and bulk find/replace to be able to get the salary into a number value in PostgreSQL. For ease of querying, we also had to "normalize" player names such that they didn't have accents, since having accents would make our queries much more complex since we all use American English keyboards. We did a few further manipulations; first, we removed duplicate statistics that were double-counted by the dataset for players who switched teams mid-season. Finally, we had to also add a new column for salaries adjusted for inflation.

References

"NBA Player Salaries." *HoopsHype*, USA Today Sports, hoopshype.com/salaries/players/. Accessed 1 Dec. 2023.

Datta, Sumitro. "NBA Stats (1947-Present)." Kaggle, 1 Nov. 2023, www.kaggle.com/datasets/sumitrodatta/nba-aba-baa-stats/.