Lab 13: Relational Databases

Statistical Computing, 36-350

Week of Tuesday November 29, 2022

Name:
Andrew ID:
Collaborated with:

This lab is to be done in class (completed outside of class time if need be). You can collaborate with your classmates, but you must identify their names above, and you must submit **your own** lab as an knitted PDF file on Gradescope, by Friday 9pm, this week.

This week's agenda: practicing SQLite queries, performing simple computations and joins, and testing our understanding by writing equivalent R code for these database manipulations.

Lahman baseball database

Thanks to Sean Lahman, extensive baseball data is freely available all the way back to the 1871 season. We're going ot use a SQLite version of the baseball database put together by Jeff Knecht, at https://github.com/jknecht/baseball-archive-sqlite. The most recent SQLite database was recently updated to include the 2016 season. It has been posted to the class website at https://www.stat.cmu.edu/~arinaldo/Teaching/36350/F22/data/lahman2016.sqlite. Download this file (it's about 50 MB) and save it in the working directory for your lab.

Q1. Practice with SQL data extraction

- **1a.** Install the packages `DBI`, `RSQLite` if you haven't done so already, and load them into your R session. Using `dbDriver()`, `dbConnect()`, set up a connection called `con` the SQLite database stored in lahman2016.sqlite. Then, use `dbListTables()` to list the tables in the database.

```
# YOUR CODE GOES HERE
```

- **1b.** Using `dbReadTable()`, grab the table named "Batting" and save it as a data frame in your R session, called `batting`. Check that `batting` is indeed a data frame, and that it has dimension 102816 x 24.

```
# YOUR CODE GOES HERE
```

- **1c.** Remove `eval=FALSE` from the preamble in the R code chunks below. Then, after each SQL query (each call to `dbGetQuery()`), explain in words what is being extracted, and write one line of base R code (sometimes you might need two lines) to get the same result using the `batting` data frame.

```
dbGetQuery(con, paste("SELECT playerID, yearID, AB, H, HR",
                      "FROM Batting",
                      "ORDER BY yearID",
                      "LIMIT 10"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT playerID, yearID, AB, H, HR",
                      "FROM Batting",
                      "ORDER BY HR DESC",
                      "LIMIT 10"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT playerID, yearID, AB, H, HR",
                      "FROM Batting",
                      "WHERE HR > 55",
                      "ORDER BY HR DESC"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT playerID, yearID, AB, H, HR",
                      "FROM Batting",
                      "WHERE yearID >= 1990 AND yearID <= 2000",
                      "ORDER BY HR DESC",
                      "LIMIT 10"))
```

```
# YOUR CODE GOES HERE
```

- **1d.** Perform the same computations in the last question, but now using `dplyr` verbs and pipes.

```
# YOUR CODE GOES HERE
```

Q2. Practice with SQL computations

- **2a.** As before, remove `eval=FALSE` from the preamble in the following R code chunks. Then, after each SQL query, explain in words what is being extracted, and write one line of base R code to get the same result using the `batting` data frame. Hint: often you'll have to use `na.rm=TRUE` to deal with NA values, for example `mean(x, na.rm=TRUE)` computes the mean of a vector `x` after removing any NA values.

```
dbGetQuery(con, paste("SELECT AVG(HR)",
                      "FROM Batting"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT SUM(HR)",
                      "FROM Batting"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT playerID, yearID, teamID, MAX(HR)",
                      "FROM Batting"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT AVG(HR)",
                      "FROM Batting",
                      "WHERE yearID >= 1990"))
```

```
# YOUR CODE GOES HERE
```

- **2b.** Again, after each SQL query explain in words what is being extracted, and write one line (or two lines) of R code to get the same result using the `batting` data frame. You may use base R, `dplyr`, pipes, or whatever means you want.

```
dbGetQuery(con, paste("SELECT teamID, AVG(HR)",
                      "FROM Batting",
```

```
                        "WHERE yearID >= 1990",
                        "GROUP BY teamID",
                        "LIMIT 5"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT teamID, AVG(HR)",
                        "FROM Batting",
                        "WHERE yearID < 1960",
                        "GROUP BY teamID",
                        "ORDER BY AVG(HR) DESC",
                        "LIMIT 5"))
```

```
# YOUR CODE GOES HERE
```

```
dbGetQuery(con, paste("SELECT teamID, yearID, AVG(HR)",
                        "FROM Batting",
                        "WHERE yearID == 1991 OR yearID == 1992",
                        "GROUP BY teamID, yearID",
                        "ORDER BY AVG(HR) DESC",
                        "LIMIT 15"))
```

```
# YOUR CODE GOES HERE
```

Q3. More practice with computations

- **3a.** Use a SQL query on the "Batting" table to calculate each player's average number of hits (`H`) over the seasons they played, and display the players with the 10 highest hit averages, along with their hit averages. Hint: `AVG()`, `GROUP BY`, `ORDER BY`.

```
# YOUR CODE GOES HERE
```

- **3b.** Calculate the same as in the last question, but now display all players whose hit averages are above 170. Hint: `HAVING`.

```
# YOUR CODE GOES HERE
```

- **3c.** Calculate the same as in the last question, but now display for all players with hit averages above 170—-in addition to the player's ID and his batting average—the last year in which each player played.

```
# YOUR CODE GOES HERE
```

Q4. Practice with SQL join operations

- **4a.** Using `JOIN`, merge the "Batting" and "Salaries" tables based on matching the `yearID`, `playerID` pairs. Display the year, player, salary, and number of hits for the first 10 records.

```
# YOUR CODE GOES HERE
```

- **4b.** Building off of the code from the end of lecture, which does something similar, compute the average salaries for the players with the top 10 highest hit averages.

```
# YOUR CODE GOES HERE
```

- **4c.** Compute the hit averages for the players with the top 10 highest salaries. Hint: this should only require a very small tweak to the code you wrote for the last question.

```
# YOUR CODE GOES HERE
```

- **4d.** Using the "Fielding" table, list the 10 worst (highest) number of errors (`E`) committed by a player in a season, only considering the year 1990 and later. In addition to the number of errors, list the year and player ID for each record.

```
# YOUR CODE GOES HERE
```

- **4e.** By appropriately merging the "Fielding" and "Salaries" tables, list the salaries for each record that you extracted in the last question. Then, answer the following question: what was the highest salary paid to a player who made at least 30 errors in a season, after 1990?

```
# YOUR CODE GOES HERE
```

Q5. All about the money

- **5a.** Use a SQL query on the "Salaries" table to compute the payroll (total of salaries) for each team in the year 2010, and display the 3 teams with the highest payrolls. Do the same, but display the 3 teams with the lowest payroll (ouch!).

```
# YOUR CODE GOES HERE
```

- **5b.** Use a SQL query to compute the total payroll for each team, added up over the years between 1985 and 2016. Hint: `dbGetQuery()` actually returns a data frame. You should have a data frame of dimension 46 x 2, and the 2 columns should display the team ID and the payroll. Check that your data frame has the right dimensions and display its first 10 rows. Then, answer: what team has the highest total payroll? The lowest payroll? Where do the Pirates rank?

```
# YOUR CODE GOES HERE
```

- **5c.** Use a SQL query to compute the payroll for each team, separately for each year in between 1985 and 2016. Hint: `GROUP BY` can take two arguments, separated by a comma. You should have a data frame of dimension 918 x 3, and the 3 columns should be display the team ID, year, and payroll. Check that your data frame has the proper dimensions, and display its last 10 rows.

```
# YOUR CODE GOES HERE
```

- **5d.** Plot the Pittsburgh Pirates' payroll over time (i.e., over the years 1985 to 2016), with appropriately labeled axes and an appropriate title. What is the trend that you see?

```
# YOUR CODE GOES HERE
```

- **Challenge.** On a single plot, display the payrolls over time (i.e., over the years 1985 to 2016) for 8 teams of your choosing. Make sure that their payroll curves are distinguishable (by color, line type, some combo, you choose). Use appropriately labeled axes, an appropriate title, and an informative legend.

```
# YOUR CODE GOES HERE
```

- **Challenge.** To make these plots more sensible, we need to adjust for inflation. Find data on the average consumer price index (CPI) over the years 1985 to 2016, and use this to adjust the payrolls for inflation and reproduce your plot from the last question. Comment on the changes.

Q6. Batting averages (optional)

- **6a.** Use a SQL query to calculate the top 10 best batting averages achieved by a player in any season after 1940. Note: batting average is the number of hits (`H`) divided by number of at bats (`AB`) achieved by a player in a given season, but (let's say) it is only defined for players that have at least 400 at bats in that season. Your resulting data frame from the SQL query should be 10 x 3, with the 3 columns displaying the playerID, yearID, and batting average.

```
# YOUR CODE GOES HERE
```

- **6b.** Compute batting averages as described above, but now plot a histogram of all of these battings averages (aggregated over all players and all seasons after 1940), with an appropriate title. Use a large value of the `breaks` argument to get a good sense of the shape of the histogram. Does this look like a normal distribution to you? What is the estimated mean and the standard deviation? Overlay the normal density curve on top of your histogram, with the appropriate mean and variance, and comment on how it fits. Perform a rigorous hypothesis test for normality of batting averages here; you might consider using `ks.test()`.

```
# YOUR CODE GOES HERE
```

- **6c.** For the computed batting averages in the last question, separate out the batting averages before and after 1985. Plot two overlaid histograms, using transparent colors, for the batting averages before and after 1985. Set an appropriate title and informative legend. Do the distributions look different? If so, how? Perform a rigorous hypothesis test for the difference in distributions here; you might again consider using `ks.test()`.

```
# YOUR CODE GOES HERE
```

- **6d.** Modifying your last SQL query so that you also extract, in addition to the batting averages, the number of home runs (for all players and all seasons after 1940). Produce a scatterplot of the number of home runs versus the batting average, with appropriate axes labels and an appropriate title. What does the general trend appear to be? Overlay the least squares regression line on top of your plot. What could go wrong with using this regression line to predict a player's home run total from their batting average?

```
# YOUR CODE GOES HERE
```