index

February 25, 2022

1 Grouping Data with SQL - Lab

1.1 Introduction

In this lab, you'll query data from a table populated with Babe Ruth's career hitting statistics. Then you'll use aggregate functions to pull interesting information from the table that basic queries cannot track.

1.2 Objectives

- Describe the relationship between aggregate functions and `GROUP BY` statements
- Use `Group BY` statements in SQL to apply aggregate functions like: `COUNT`, `MAX`, `MIN`, and `SUM`
- Create an alias in a SQL query
- Use the `HAVING` clause to compare different aggregates
- Compare the difference between the `WHERE` and `HAVING` clause

1.3 Babe Ruth - Career Hitting Statistics

The database you will be working with in this lab is located in the file `babe_ruth.db`. This database contains a single table, `babe_ruth_stats`. The table schema is:

```
CREATE TABLE babe_ruth_stats (
  id INTEGER PRIMARY KEY,
  year INTEGER,
  team TEXT,
  league TEXT,
  doubles INTEGER,
  triples INTEGER,
  hits INTEGER,
  HR INTEGER,
  games INTEGER,
  runs INTEGER,
  RBI INTEGER,
  at_bats INTEGER,
  BB INTEGER,
  SB INTEGER,
  SO INTEGER,
  AVG REAL
)
```

The table contains the following data:

year	team	league	doubles	triples	hits	HR	games	runs	RBI	at_bats	BB	SB	SO	AVG
1914	"BOS"	"AL"	1	0	2	0	5	1	2	10	0	0	4	0.2
1915	"BOS"	"AL"	10	1	29	4	42	16	21	92	9	0	23	0.315
1916	"BOS"	"AL"	5	3	37	3	67	18	15	136	10	0	23	0.272
1917	"BOS"	"AL"	6	3	40	2	52	14	12	123	12	0	18	0.325
1918	"BOS"	"AL"	26	11	95	11	95	50	66	317	58	6	58	0.3
1919	"BOS"	"AL"	34	12	139	29	130	103	114	432	101	7	58	0.322
1920	"NY"	"AL"	36	9	172	54	142	158	137	458	150	14	80	0.376
1921	"NY"	"AL"	44	16	204	59	152	177	171	540	145	17	81	0.378
1922	"NY"	"AL"	24	8	128	35	110	94	99	406	84	2	80	0.315
1923	"NY"	"AL"	45	13	205	41	152	151	131	522	170	17	93	0.393
1924	"NY"	"AL"	39	7	200	46	153	143	121	529	142	9	81	0.378
1925	"NY"	"AL"	12	2	104	25	98	61	66	359	59	2	68	0.29
1926	"NY"	"AL"	30	5	184	47	152	139	146	495	144	11	76	0.372
1927	"NY"	"AL"	29	8	192	60	151	158	164	540	137	7	89	0.356
1928	"NY"	"AL"	29	8	173	54	154	163	142	536	137	4	87	0.323
1929	"NY"	"AL"	26	6	172	46	135	121	154	499	72	5	60	0.345
1930	"NY"	"AL"	28	9	186	49	145	150	153	518	136	10	61	0.359
1931	"NY"	"AL"	31	3	199	46	145	149	163	534	128	5	51	0.373
1932	"NY"	"AL"	13	5	156	41	133	120	137	457	130	2	62	0.341
1933	"NY"	"AL"	21	3	138	34	137	97	103	459	114	4	90	0.301
1934	"NY"	"AL"	17	4	105	22	125	78	84	365	104	1	63	0.288
1935	"BOS"	"NL"	0	0	13	6	28	13	12	72	20	0	24	0.181

As you can see, each record in this table represents statistics for a baseball season.

1.4 Connect to the Database

Import `sqlite3` and `pandas`. Then, connect to the database in the `babe_ruth.db` file.

[2]:
```python
# Your code here
import sqlite3
import pandas as pd

conn = sqlite3.connect("babe_ruth.db")
```

Now, write SQL queries to answer questions about the data in the `babe_ruth_stats` table. You can display all results using pandas for readability.

1.5 Total Seasons

Return the total number of years that Babe Ruth played professional baseball

[3]:
```python
# Your code here
years = """
```

```
SELECT COUNT(*) as Num_Seasons
FROM babe_ruth_stats;
"""

pd.read_sql(years, conn)
```

[3]:
```
   Num_Seasons
0           22
```

1.6 Seasons with NY

Return the total number of years Babe Ruth played with the NY Yankees (i.e. where the `team` value is `"NY"`).

[7]:
```
# Your code here
years_NY = """
SELECT year, COUNT(*) as Num_of_NY_GAMES
FROM babe_ruth_stats
WHERE team = "NY"
GROUP BY year;
"""
pd.read_sql(years_NY, conn)
```

[7]:
```
    year  Num_of_NY_GAMES
0   1920                1
1   1921                1
2   1922                1
3   1923                1
4   1924                1
5   1925                1
6   1926                1
7   1927                1
8   1928                1
9   1929                1
10  1930                1
11  1931                1
12  1932                1
13  1933                1
14  1934                1
```

[8]:
```
# Your code here
years_NY = """
SELECT year, COUNT(*) as Num_of_NY_GAMES
FROM babe_ruth_stats
WHERE team = "NY"
GROUP BY year;
"""
```

```
pd.read_sql(years_NY, conn)
```

```
[8]:     year  Num_of_NY_GAMES
     0   1920                1
     1   1921                1
     2   1922                1
     3   1923                1
     4   1924                1
     5   1925                1
     6   1926                1
     7   1927                1
     8   1928                1
     9   1929                1
     10  1930                1
     11  1931                1
     12  1932                1
     13  1933                1
     14  1934                1
```

1.7 Most Home Runs

Return the row with the most HR that Babe Ruth hit in one season.

```
[9]: # Your code here
     max_hr = """
     SELECT *
     FROM babe_ruth_stats
     ORDER BY HR DESC
     LIMIT 1;
     """
     pd.read_sql(max_hr, conn)
```

```
[9]:    id  year team league  doubles  triples  hits  HR  games  runs  RBI  \
     0  14  1927   NY     AL       29        8   192  60    151   158  164

        at_bats   BB  SB  SO    AVG
     0      540  137   7  89  0.356
```

```
[10]: # From GitHub

      q = """
      SELECT *
      FROM babe_ruth_stats
      WHERE HR = (
          SELECT MAX(HR)
          FROM babe_ruth_stats
      )
```

```
;
"""
pd.read_sql(q, conn)
```

```
[10]:    id  year team league  doubles  triples  hits  HR  games  runs  RBI  \
      0  14  1927   NY     AL       29        8   192  60    151   158  164

         at_bats   BB  SB  SO    AVG
      0      540  137   7  89  0.356
```

1.8 Least HR

Select the row with the least number of HR hit in one season.

```
[11]: # Your code here
      min_hr = """
      SELECT *
      FROM babe_ruth_stats
      ORDER BY HR ASC
      LIMIT 1;
      """
      pd.read_sql(min_hr, conn)
```

```
[11]:    id  year team league  doubles  triples  hits  HR  games  runs  RBI  \
      0   1  1914  BOS     AL        1        0     2   0      5     1    2

         at_bats  BB  SB  SO  AVG
      0       10   0   0   4  0.2
```

```
[12]: # From GitHub

      q = """
      SELECT *
      FROM babe_ruth_stats
      WHERE HR = (
          SELECT MIN(HR)
          FROM babe_ruth_stats
      )
      ;
      """
      pd.read_sql(q, conn)
```

```
[12]:    id  year team league  doubles  triples  hits  HR  games  runs  RBI  \
      0   1  1914  BOS     AL        1        0     2   0      5     1    2

         at_bats  BB  SB  SO  AVG
      0       10   0   0   4  0.2
```

1.9 Total HR

Return the total number of HR hit by Babe Ruth during his career.

```
[13]: # Your code here
      total_HR = """
      SELECT SUM(HR) as total_HR
      FROM babe_ruth_stats
      """
      pd.read_sql(total_HR,conn)
```

```
[13]:    total_HR
      0       714
```

1.10 Five Worst HR Seasons With at Least 100 Games Played

Above you saw that Babe Ruth hit 0 home runs in his first year when he played only five games. To avoid this and other extreme outliers, first filter the data to include only those years in which Ruth played in at least 100 games. Then, select all of the columns for the 5 worst seasons, in terms of the number of home runs, where he played over 100 games.

```
[14]: # Your code here
      Worst = """
      SELECT *
      FROM babe_ruth_stats
      WHERE games > 100
      ORDER BY HR ASC
      LIMIT 5;
      """
      pd.read_sql(Worst,conn)
```

```
[14]:    id  year team league  doubles  triples  hits  HR  games  runs  RBI  \
      0  21  1934   NY     AL       17        4   105  22    125    78   84
      1   6  1919  BOS     AL       34       12   139  29    130   103  114
      2  20  1933   NY     AL       21        3   138  34    137    97  103
      3   9  1922   NY     AL       24        8   128  35    110    94   99
      4  10  1923   NY     AL       45       13   205  41    152   151  131

         at_bats   BB  SB  SO    AVG
      0      365  104   1  63  0.288
      1      432  101   7  58  0.322
      2      459  114   4  90  0.301
      3      406   84   2  80  0.315
      4      522  170  17  93  0.393
```

1.11 Average Batting Average

Select the average, `AVG`, of Ruth's batting averages. The header of the result would be `AVG(AVG)` which is quite confusing, so provide an alias of `career_average`.

```
[15]: # Your code here
      ave = """
      SELECT AVG(AVG) AS career_average
      FROM babe_ruth_stats
      """
      pd.read_sql(ave,conn)
```

```
[15]:    career_average
      0        0.322864
```

1.12 Number of Years with Over 300 Times On Base

We want to know the years in which Ruth successfully reached base over 300 times. We need to add `hits` and `BB` to calculate how many times Ruth reached base. Simply add the two columns together (ie: `SELECT [columnName] + [columnName] AS ...`) and give this value an alias of `on_base`. Select the `year` and `on_base` for only those years with an `on_base` over 300.

```
[16]: # Your code here
      years300 = """
      SELECT year, hits + BB AS on_base
      FROM babe_ruth_stats
      WHERE on_base > 300
      ORDER BY on_base DESC
      """
      pd.read_sql(years300,conn)
```

```
[16]:    year  on_base
      0  1923      375
      1  1921      349
      2  1924      342
      3  1927      329
      4  1926      328
      5  1931      327
      6  1920      322
      7  1930      322
      8  1928      310
```

1.13 Total Years and Hits Per Team

Select the total number of years played (as `num_seasons`) and total hits (as `total_hits`) Babe Ruth had for each team he played for. The result should have 2 rows, one for each team.

```
[17]: # Your code here
      hit_per_team = """
      SELECT team, COUNT(*) AS num_seasosn, SUM(hits) AS total_hits
      FROM babe_ruth_stats
      GROUP BY team;
```

```
"""
pd.read_sql(hit_per_team, conn)
```

[17]:
```
  team  num_seasosn  total_hits
0  BOS            7         355
1   NY           15        2518
```

1.14 Teams with More than 10 Seasons

Repeat the above query, this time only including teams where he played for more than 10 years.

Hint: Think about whether this filtering occurs before or after the `GROUP BY`. If before, that's a `WHERE`. If after, that's a `HAVING`.

[70]:
```
# Your code here
team10 = """
SELECT team, COUNT(*) AS num_seasons, SUM(hits) AS total_hits
FROM babe_ruth_stats
GROUP BY team
HAVING num_seasons > 10
"""

pd.read_sql(team10, conn)
```

[70]:
```
  team  num_seasons  total_hits
0   NY           15        2518
```

1.15 Team with Highest Average At Bats

Select the name of the team and the average at bats per season (as `average_at_bats`), for the team where he averaged the highest at bats.

[78]:
```
# Your code here
Havg = """
SELECT team, AVG(at_bats) AS average_at_bats
FROM babe_ruth_stats
GROUP BY team
ORDER BY AVG(at_bats) DESC
"""

pd.read_sql(Havg, conn)
```

[78]:
```
  team  average_at_bats
0   NY       481.133333
1  BOS       168.857143
```

[75]:
```

```

```
[75]:   team  average_at_bats
      0  BOS       168.857143
      1   NY       481.133333
```

1.16 Teams with Average At Bats Over 100

Repeat the above query, this time returning all teams where the `average_at_bats` was over 100.

```
[77]: # Your code here
      ave100 = """
      SELECT team, AVG(at_bats) AS average_at_bats
      FROM babe_ruth_stats
      GROUP BY team
      HAVING AVG(at_bats) > 100
      """

      pd.read_sql(ave100, conn)
```

```
[77]:   team  average_at_bats
      0  BOS       168.857143
      1   NY       481.133333
```

1.17 Summary

Well done! In this lab, you continued to add complexity to SQL statements, which included using some aggregate functions, the `GROUP BY` statement, and the `HAVING` statement. You wrote queries that showed Babe Ruth's total years and home runs per team as well as selected only years that met a minimum value of our calculated on base attribute.

```
[ ]: # Grouping Data with SQL - Lab

     ## Introduction

     In this lab, you'll query data from a table populated with Babe Ruth's career␣
      ↪hitting statistics.  Then you'll use aggregate functions to pull interesting␣
      ↪information from the table that basic queries cannot track.

     ## Objectives

     * Describe the relationship between aggregate functions and `GROUP BY`␣
      ↪statements
     * Use `Group BY` statements in SQL to apply aggregate functions like: `COUNT`,␣
      ↪`MAX`, `MIN`, and `SUM`
     * Create an alias in a SQL query
     * Use the `HAVING` clause to compare different aggregates
     * Compare the difference between the `WHERE` and `HAVING` clause
```

~~~~
## Babe Ruth - Career Hitting Statistics

The database you will be working with in this lab is located in the file `babe_ruth.db`. This database contains a single table, `babe_ruth_stats`. The table schema is:

```
CREATE TABLE babe_ruth_stats (
  id INTEGER PRIMARY KEY,
  year INTEGER,
  team TEXT,
  league TEXT,
  doubles INTEGER,
  triples INTEGER,
  hits INTEGER,
  HR INTEGER,
  games INTEGER,
  runs INTEGER,
  RBI INTEGER,
  at_bats INTEGER,
  BB INTEGER,
  SB INTEGER,
  SO INTEGER,
  AVG REAL
)
```

The table contains the following data:

year|team |league|doubles|triples|hits|HR|games|runs|RBI|at_bats|BB |SB|SO|AVG
----|-----|------|-------|-------|----|--|-----|----|---|-------|---|--|--|------
1914|"BOS"|"AL"  |1      |0      |2   |0 |5    |1   |2  |10     |0  |0 |4 |0.2
1915|"BOS"|"AL"  |10     |1      |29  |4 |42   |16  |21 |92     |9  |0 |23|0.315
1916|"BOS"|"AL"  |5      |3      |37  |3 |67   |18  |15 |136    |10 |0 |23|0.272
1917|"BOS"|"AL"  |6      |3      |40  |2 |52   |14  |12 |123    |12 |0 |18|0.325
1918|"BOS"|"AL"  |26     |11     |95  |11|95   |50  |66 |317    |58 |6 |58|0.3
1919|"BOS"|"AL"  |34     |12     |139 |29|130  |103 |114|432    |101|7 |58|0.322
1920|"NY" |"AL"  |36     |9      |172 |54|142  |158 |137|458    |150|14|80|0.376
1921|"NY" |"AL"  |44     |16     |204 |59|152  |177 |171|540    |145|17|81|0.378
1922|"NY" |"AL"  |24     |8      |128 |35|110  |94  |99 |406    |84 |2 |80|0.315
1923|"NY" |"AL"  |45     |13     |205 |41|152  |151 |131|522    |170|17|93|0.393
1924|"NY" |"AL"  |39     |7      |200 |46|153  |143 |121|529    |142|9 |81|0.378
1925|"NY" |"AL"  |12     |2      |104 |25|98   |61  |66 |359    |59 |2 |68|0.29
1926|"NY" |"AL"  |30     |5      |184 |47|152  |139 |146|495    |144|11|76|0.372
1927|"NY" |"AL"  |29     |8      |192 |60|151  |158 |164|540    |137|7 |89|0.356
1928|"NY" |"AL"  |29     |8      |173 |54|154  |163 |142|536    |137|4 |87|0.323
~~~~

```
1929|"NY" |"AL"  |26     |6      |172 |46|135  |121 |154|499    |72 |5 |60|0.345
1930|"NY" |"AL"  |28     |9      |186 |49|145  |150 |153|518    |136|10|61|0.359
1931|"NY" |"AL"  |31     |3      |199 |46|145  |149 |163|534    |128|5 |51|0.373
1932|"NY" |"AL"  |13     |5      |156 |41|133  |120 |137|457    |130|2 |62|0.341
1933|"NY" |"AL"  |21     |3      |138 |34|137  |97  |103|459    |114|4 |90|0.301
1934|"NY" |"AL"  |17     |4      |105 |22|125  |78  |84 |365    |104|1 |63|0.288
1935|"BOS"|"NL"  |0      |0      |13  |6 |28   |13  |12 |72     |20 |0 |24|0.181

As you can see, each record in this table represents statistics for a baseball␣
 ↪season.

## Connect to the Database

Import `sqlite3` and `pandas`. Then, connect to the database in the `babe_ruth.
 ↪db` file.

# Your code here
import sqlite3
import pandas as pd

conn = sqlite3.connect("babe_ruth.db")

Now, write SQL queries to answer questions about the data in the␣
 ↪`babe_ruth_stats` table. You can display all results using pandas for␣
 ↪readability.

## Total Seasons
Return the total number of years that Babe Ruth played professional baseball

# Your code here
years = """
SELECT COUNT(*) as Num_Seasons
FROM babe_ruth_stats;
"""

pd.read_sql(years, conn)

## Seasons with NY
Return the total number of years Babe Ruth played with the NY Yankees (i.e.␣
 ↪where the `team` value is `"NY"`).

# Your code here
years_NY = """
SELECT year, COUNT(*) as Num_of_NY_GAMES
FROM babe_ruth_stats
WHERE team = "NY"
```

```
GROUP BY year;
"""
pd.read_sql(years_NY, conn)

# Your code here
years_NY = """
SELECT year, COUNT(*) as Num_of_NY_GAMES
FROM babe_ruth_stats
WHERE team = "NY"
GROUP BY year;
"""
pd.read_sql(years_NY, conn)

## Most Home Runs

Return the row with the most HR that Babe Ruth hit in one season.

# Your code here
max_hr = """
SELECT *
FROM babe_ruth_stats
ORDER BY HR DESC
LIMIT 1;
"""
pd.read_sql(max_hr, conn)

# From GitHub

q = """
SELECT *
FROM babe_ruth_stats
WHERE HR = (
    SELECT MAX(HR)
    FROM babe_ruth_stats
)
;
"""
pd.read_sql(q, conn)

## Least HR
Select the row with the least number of HR hit in one season.

# Your code here
min_hr = """
SELECT *
FROM babe_ruth_stats
ORDER BY HR ASC
```

```
LIMIT 1;
"""
pd.read_sql(min_hr, conn)

# From GitHub

q = """
SELECT *
FROM babe_ruth_stats
WHERE HR = (
    SELECT MIN(HR)
    FROM babe_ruth_stats
)
;
"""
pd.read_sql(q, conn)

## Total HR
Return the total number of HR hit by Babe Ruth during his career.

# Your code here
total_HR = """
SELECT SUM(HR) as total_HR
FROM babe_ruth_stats
"""
pd.read_sql(total_HR,conn)

##  Five Worst HR Seasons With at Least 100 Games Played
Above you saw that Babe Ruth hit 0 home runs in his first year when he played␣
 ↪only five games.  To avoid this and other extreme  outliers, first filter␣
 ↪the data to include only those years in which Ruth played in at least 100␣
 ↪games. Then, select all of the columns for the 5 worst seasons, in terms of␣
 ↪the number of home runs, where he played over 100 games.

# Your code here
Worst = """
SELECT *
FROM babe_ruth_stats
WHERE games > 100
ORDER BY HR ASC
LIMIT 5;
"""
pd.read_sql(Worst,conn)

## Average Batting Average
```

```
Select the average, `AVG`, of Ruth's batting averages.  The header of the␣
  ↪result would be `AVG(AVG)` which is quite confusing, so provide an alias of␣
  ↪`career_average`.

# Your code here
ave = """
SELECT AVG(AVG) AS career_average
FROM babe_ruth_stats
"""
pd.read_sql(ave,conn)

## Number of Years with Over 300 Times On Base
We want to know the years in which Ruth successfully reached base over 300␣
  ↪times.  We need to add `hits` and `BB` to calculate how many times Ruth␣
  ↪reached base.  Simply add the two columns together (ie: `SELECT [columnName]␣
  ↪+ [columnName] AS ...`) and give this value an alias of `on_base`.  Select␣
  ↪the `year` and `on_base` for only those years with an `on_base` over 300.

# Your code here
years300 = """
SELECT year, hits + BB AS on_base
FROM babe_ruth_stats
WHERE on_base > 300
ORDER BY on_base DESC
"""
pd.read_sql(years300,conn)

## Total Years and Hits Per Team
Select the total number of years played (as `num_seasons`) and total hits (as␣
  ↪`total_hits`) Babe Ruth had for each team he played for. The result should␣
  ↪have 2 rows, one for each team.

# Your code here
hit_per_team = """
SELECT team, COUNT(*) AS num_seasosn, SUM(hits) AS total_hits
FROM babe_ruth_stats
GROUP BY team;
"""
pd.read_sql(hit_per_team, conn)

## Teams with More than 10 Seasons
Repeat the above query, this time only including teams where he played for more␣
  ↪than 10 years.
```

```
**Hint:** Think about whether this filtering occurs before or after the `GROUP␣
↪BY`. If before, that's a `WHERE`. If after, that's a `HAVING`.

# Your code here
team10 = """
SELECT team, COUNT(*) AS num_seasons, SUM(hits) AS total_hits
FROM babe_ruth_stats
GROUP BY team
HAVING num_seasons > 10
"""

pd.read_sql(team10, conn)

## Team with Highest Average At Bats

Select the name of the team and the average at bats per season (as␣
↪`average_at_bats`), for the team where he averaged the highest at bats.

# Your code here
Havg = """
SELECT team, AVG(at_bats) AS average_at_bats
FROM babe_ruth_stats
GROUP BY team
ORDER BY AVG(at_bats) DESC
"""

pd.read_sql(Havg, conn)



## Teams with Average At Bats Over 100

Repeat the above query, this time returning all teams where the␣
↪`average_at_bats` was over 100.

# Your code here
ave100 = """
SELECT team, AVG(at_bats) AS average_at_bats
FROM babe_ruth_stats
GROUP BY team
HAVING AVG(at_bats) > 100
"""

pd.read_sql(ave100, conn)

## Summary
```

```
Well done! In this lab, you continued to add complexity to SQL statements,␣
 ↪which included using some aggregate functions, the `GROUP BY` statement, and␣
 ↪the `HAVING` statement. You wrote queries that showed Babe Ruth's total␣
 ↪years and home runs per team as well as selected only years that met a␣
 ↪minimum value of our calculated on base attribute.
```