learn-co-curriculum / dsc-grouping-data-with-sql-lab Public

View license

2 stars 199 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 13 commits ahead, 11 commits behind master. Contribute

hoffm386 pd.read_sql, augment HAVING practice … on Apr 14, 2021 18

View code

README.md

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
1914	"BOS"	"AL"	1	0	2	0	5	1	2
1915	"BOS"	"AL"	10	1	29	4	42	16	21
1916	"BOS"	"AL"	5	3	37	3	67	18	15
1917	"BOS"	"AL"	6	3	40	2	52	14	12
1918	"BOS"	"AL"	26	11	95	11	95	50	66
1919	"BOS"	"AL"	34	12	139	29	130	103	114
1920	"NY"	"AL"	36	9	172	54	142	158	137
1921	"NY"	"AL"	44	16	204	59	152	177	171
1922	"NY"	"AL"	24	8	128	35	110	94	99
1923	"NY"	"AL"	45	13	205	41	152	151	131

year	team	league	doubles	triples	hits	HR	games	runs	RBI
1924	"NY"	"AL"	39	7	200	46	153	143	121
1925	"NY"	"AL"	12	2	104	25	98	61	66
1926	"NY"	"AL"	30	5	184	47	152	139	146
1927	"NY"	"AL"	29	8	192	60	151	158	164
1928	"NY"	"AL"	29	8	173	54	154	163	142
1929	"NY"	"AL"	26	6	172	46	135	121	154
1930	"NY"	"AL"	28	9	186	49	145	150	153
1931	"NY"	"AL"	31	3	199	46	145	149	163
1932	"NY"	"AL"	13	5	156	41	133	120	137
1933	"NY"	"AL"	21	3	138	34	137	97	103
1934	"NY"	"AL"	17	4	105	22	125	78	84
1935	"BOS"	"NL"	0	0	13	6	28	13	12

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.

```
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

```
# Note that we don't need to group by anything since every
# record in the table represents a year. We can just go
# right to counting records
q = """
SELECT COUNT(*) AS num_seasons
FROM babe_ruth_stats
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	num_seasons
0	22

Seasons with NY

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

```
q = """
SELECT COUNT(*) as num_seasons_ny
FROM babe_ruth_stats
WHERE team = "NY"
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

</style>

	num_seasons_ny
0	15

Most Home Runs

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

```
q = """
SELECT *
FROM babe_ruth_stats
ORDER BY HR DESC
LIMIT 1
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	year	team	league	doubles	triples	hits	HR	games	r
0	14	1927	NY	AL	29	8	192	60	151	1

```
# Alternatively, one could also write the following query in order to
# use the MAX function instead of ORDER BY and LIMIT
# This includes a subquery, which you will see in an upcoming lesson:
q = """
```

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

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	year	team	league	doubles	triples	hits	HR	games	r
0	14	1927	NY	AL	29	8	192	60	151	1

Least HR

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

```
# This is the same as the previous query, without DESC
q = """
SELECT *
FROM babe_ruth_stats
ORDER BY HR
LIMIT 1
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

</style>

	id	year	team	league	doubles	triples	hits	HR	games	ru
0	1	1914	BOS	AL	1	0	2	0	5	1

```
# Again, there is a way to use a subquery and MIN
q = """
SELECT *
FROM babe_ruth_stats
WHERE HR = (
    SELECT MIN(HR)
    FROM babe_ruth_stats
)
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	year	team	league	doubles	triples	hits	HR	games	ru
0	1	1914	BOS	AL	1	0	2	0	5	1

Total HR

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

```
q = """
SELECT SUM(HR) AS total_home_runs
FROM babe_ruth_stats
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	total_home_runs
0	714

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.

```
q = """
SELECT *
FROM babe_ruth_stats
WHERE games > 100
ORDER BY HR
LIMIT 5
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
  }

  .dataframe thead th {
      text-align: right;
  }
```

</style>

	id	year	team	league	doubles	triples	hits	HR	games	r
0	21	1934	NY	AL	17	4	105	22	125	7
1	6	1919	BOS	AL	34	12	139	29	130	1
2	20	1933	NY	AL	21	3	138	34	137	9
3	9	1922	NY	AL	24	8	128	35	110	9
4	10	1923	NY	AL	45	13	205	41	152	1

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`.

```
q = """
SELECT AVG(AVG) AS career_average
FROM babe_ruth_stats
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
  .dataframe tbody tr th {
      vertical-align: top;
  }

  .dataframe thead th {
      text-align: right;
  }
```

</style>

	career_average

	career_average
0	0.322864

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.

```
q = """
SELECT year, hits + BB AS on_base
FROM babe_ruth_stats
WHERE on_base > 300
ORDER BY on_base DESC
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	year	on_base
0	1923	375
1	1921	349
2	1924	342
3	1927	329
4	1926	328

	year	on_base
5	1931	327
6	1920	322
7	1930	322
8	1928	310

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.

```
q = """
SELECT team, COUNT(*) AS num_seasons, SUM(hits) AS total_hits
FROM babe_ruth_stats
GROUP BY team
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	team	num_seasons	total_hits
0	BOS	7	355
1	NY	15	2518

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`.

```
q = """
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(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	team	num_seasons	total_hits
0	NY	15	2518

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.

```
q = """
SELECT team, AVG(at_bats) AS average_at_bats
FROM babe_ruth_stats
GROUP BY team
ORDER BY average_at_bats DESC
LIMIT 1
;
```

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

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	team	average_at_bats
0	NY	481.133333

Teams with Average At Bats Over 100

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

```
q = """
SELECT team, AVG(at_bats) AS average_at_bats
FROM babe_ruth_stats
GROUP BY team
HAVING average_at_bats > 100
;
"""
pd.read_sql(q, conn)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	team	average_at_bats
0	BOS	168.857143
1	NY	481.133333

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.