HW: Week 14

36-350 – Statistical Computing

Week 14 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

You must submit **your own** lab as a knitted PDF file on Gradescope.

In this homework, you will be working with National Basketball Association (NBA) player data. In the `Files` hierarchy on Canvas, under `DATA`, you will find two files: `README` and `player-stats.sql`. The former defines the table columns that appear in the latter. It is provided for completeness; you should not have to refer to it.

Question 1

(8 points)

Download `player-stats.sql` and read it into your `postgres` session using the `\i` command. Show that you have changed the working directory inside `postgres` to where your downloaded data are stored. As a final step, display the current list of tables in your database.

```
postgres=# \cd 'C:/Users/Kimberly Zhang/Documents/36350/Week14'
postgres=# \i player-stats.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
COPY 476
COPY 476
COPY 476
postgres=# \d
                  List of relations
 Schema |           Name           |   Type   |  Owner
--------+--------------------------+----------+----------
 public | galaxies                 | table    | postgres
 public | more_player_stats        | table    | postgres
 public | more_player_stats_id_seq | sequence | postgres
 public | player_bios              | table    | postgres
 public | player_bios_id_seq       | sequence | postgres
 public | player_stats             | table    | postgres
 public | player_stats_id_seq      | sequence | postgres
(7 rows)
```

Question 2

(8 points)

Where did Danny Green go to college? Use `select` and `where` to display just that information, nothing more. (To find relevant columns of information, use `\d [TABLE NAME]` to get the list of column names.)

```
postgres=# select college
from player_bios
where firstname = 'Danny'
and lastname = 'Green';
     college
-----------------
 North Carolina
(1 row)
```

Question 3

(8 points)

Use an aggregate function from Notes 14B to determine how many unique colleges there are among the 476 players in the database. (You might want to look up the documentation for `distinct`, which is `SQL`'s analogue to `R`'s `unique()`.)

```
postgres=# select count(distinct college)
from player_bios;
 count
-------
   125
(1 row)
```

Question 4

(10 points)

Create a table called `teams`, which will have five columns: an `id` column of type `char(3)` that will be the table's primary key; `location` and `name` columns of type `text` (e.g., Boston for location, Celtics for name); a column showing the `division` in which the team plays (e.g., Atlantic); and a column showing the `conference` the team plays in (e.g., Eastern). Note that `division` and `conference` should be of type `DivisionType` and `ConferenceType` respectively; these types should be created as enumerated variables. (See the `postgres` documentation for `CREATE TYPE` and look specifically for the variant that contains the word `ENUM`; essentially, you are defining your own factor variables, with defined levels.)

```
postgres=# create type DivisionType AS ENUM ('Atlantic', 'Central', 'Southeast', 'Northwest', 'Pacific'
CREATE TYPE
postgres=# create type ConferenceType AS ENUM ('Eastern', 'Western');
CREATE TYPE
postgres=# create table teams (id char(3) primary key, location text, name text, division DivisionType,
CREATE TABLE
```

Run the following code in your `postgres()` session to populate the `teams` table:

```
insert into teams VALUES
  ('BOS', 'Boston', 'Celtics', 'Atlantic', 'Eastern'),
  ('BKN', 'Brooklyn', 'Nets', 'Atlantic', 'Eastern'),
  ('NYK', 'New York', 'Knicks', 'Atlantic', 'Eastern'),
  ('PHI', 'Philadelphia', '76ers', 'Atlantic', 'Eastern'),
```

```
('TOR', 'Toronto', 'Raptors', 'Atlantic', 'Eastern'),
('CHI', 'Chicago', 'Bulls', 'Central', 'Eastern'),
('CLE', 'Cleveland', 'Cavaliers', 'Central', 'Eastern'),
('DET', 'Detroit', 'Pistons', 'Central', 'Eastern'),
('IND', 'Indiana', 'Pacers', 'Central', 'Eastern'),
('MIL', 'Milwaukee', 'Bucks', 'Central', 'Eastern'),
('ATL', 'Atlanta', 'Hawks', 'Southeast', 'Eastern'),
('CHA', 'Charlotte', 'Bobcats', 'Southeast', 'Eastern'),
('MIA', 'Miami', 'Heat', 'Southeast', 'Eastern'),
('ORL', 'Orlando', 'Magic', 'Southeast', 'Eastern'),
('WAS', 'Washington', 'Wizards', 'Southeast', 'Eastern'),
('DEN', 'Denver', 'Nuggets', 'Northwest', 'Western'),
('MIN', 'Minnesota', 'Timberwolves', 'Northwest', 'Western'),
('OKC', 'Oklahoma City', 'Thunder', 'Northwest', 'Western'),
('POR', 'Portland', 'Trail Blazers', 'Northwest', 'Western'),
('UTA', 'Utah', 'Jazz', 'Northwest', 'Western'),
('GSW', 'Golden State', 'Warriors', 'Pacific', 'Western'),
('LAC', 'Los Angeles', 'Clippers', 'Pacific', 'Western'),
('LAL', 'Los Angeles', 'Lakers', 'Pacific', 'Western'),
('PHX', 'Phoenix', 'Suns', 'Pacific', 'Western'),
('SAC', 'Sacramento', 'Kings', 'Pacific', 'Western'),
('DAL', 'Dallas', 'Mavericks', 'Southwest', 'Western'),
('HOU', 'Houston', 'Rockets', 'Southwest', 'Western'),
('MEM', 'Memphis', 'Grizzlies', 'Southwest', 'Western'),
('NOP', 'New Orleans', 'Hornets', 'Southwest', 'Western'),
('SAS', 'San Antonio', 'Spurs', 'Southwest', 'Western');
```

Question 5

(10 points)

None of the available datasets list player positions. In the NBA, there are five commonly acknowledged positions: center, power forward, small forward, shooting guard, and point guard. One can map information in the `more_player_stats` table to these positions via the empirical formula

```
prl = per - 67*va/(gp*minutes)
```

where ranges of `prl` map directly to positions (see below). Create a new table (just call it `new_table`) that has three columns: `player`, of type `integer references more_player_stats` (it's a foreign key, something we haven't explicitly mentioned yet, but is useful for joining tables as we will see next week); `prl`, of type `numeric`; and `position`, of type `text`. Use `insert` to populate this table: you'd `select` the `id` from `more_player_stats`, and you'd select the equation above. That sounds a bit weird. Let's say table `foo` has three columns, `id`, `x`, and `y`. Now you create `bar`, that has `new_id` and `z`, which is defined as `x/y`.

- `insert into bar (new_id,z) (select id,round(x/y,1) from foo);`

(Here I decided to round off the quotient. You should do the same with `prl`.) The last thing to do is to `update` your new table by setting the position on the basis of the following criteria: `'PF'` where `prl` is greater than or equal to 11.3; `'PG'` where `prl` is [10.8,11.3); `'C'` for [10.6,10.8); and `'SG/SF'` for [0,10.6). Display the first 10 rows of your new table.

```
postgres=# create table new_table (player integer references more_player_stats, prl numeric, position t
CREATE TABLE
postgres=# insert into new_table (player, prl) (select id, round(per - 67*va/(gp*minutes),1) from more_
INSERT 0 476
postgres=# update new_table set position = (case when prl >= 11.3 then 'PF' when prl >= 10.8 and prl <
UPDATE 476
```

```
postgres=# select * from new_table limit 10;
 player | prl  | position
--------+------+----------
      1 | 11.0 | PG
      2 | 11.5 | PF
      3 | 10.5 | SG/SF
      4 | 11.5 | PF
      5 | 10.6 | C
      6 | 10.6 | C
      7 | 10.5 | SG/SF
      8 | 10.5 | SG/SF
      9 | 10.6 | C
     10 | 10.5 | SG/SF
(10 rows)
```

Question 6

(8 points)

Take the `position` column you've just defined and add it to the `player_bios` table. (Remember: `alter table` and `update`.)

You'll take advantage of the fact that `player` in `new_table` references `id` in `more_player_stats`... which means it references `id` in `player_bios` as well. After you are done, display the first five rows of `player_bios` (selecting just the `firstname`, `lastname`, and `position` columns).

```
postgres=# alter table player_bios add column position text;
ALTER TABLE
postgres=# update player_bios set position = (select position from new_table where player = id);
UPDATE 476
postgres=# select firstname, lastname, position from player_bios limit 5;
 firstname | lastname | position
-----------+----------+----------
 Aaron     | Brooks   | PG
 Aaron     | Gordon   | PF
 Aaron     | Harrison | SG/SF
 Adreian   | Payne    | PF
 Al        | Horford  | C
(5 rows)
```

Question 7

(8 points)

Now we will convert the heights given in `player_bios` from feet-inches format to simply inches. This involves `alter`ing the `player_bios` table to add a new column of type `numeric`, then using `update` to set the values of the height in inches. You may wish to use the following:

```
12*split_part(height,'-',1)::integer + split_part(height,'-',2)::integer
```

This splits the height string on '-'; the first output is feet, so we multiply that by 12 (and cast to `integer`), and the second output is inches, which is simply cast to `integer`. Finally, use two `alter` commands to drop the old height column and to rename your new column 'height'. As you did in the last question, display the first five rows of `player_bios` (but this time: `firstname`, `lastname`, and `height`).

```
postgres=# update player_bios set height_in = 12*split_part(height,'-',1)::integer + split_part(height,
UPDATE 476
postgres=# alter table player_bios drop column height;
```

```
ALTER TABLE
postgres=# alter table player_bios rename column height_in to height;
ALTER TABLE
postgres=# select firstname, lastname, height from player_bios limit 5;
 firstname | lastname  | height
-----------+-----------+--------
 Aaron     | Harrison  |     78
 Adreian   | Payne     |     82
 Al        | Horford   |     82
 Al        | Jefferson |     82
 Al-Farouq | Aminu     |     81
(5 rows)
```

For the next few questions, you will utilize the table `galaxies` that you used in Lab 14. If this is still in your `postgres` list of relations (what you see output when you do `\d`), then you are ready to go... otherwise, recreate that table and copy contents into it.

Question 8

(8 points)

Output the observed Gini coefficient, and the percentage difference of the observed Gini coefficient from the average Gini coefficient. We will define this difference to be

$$100\left(\frac{x-\bar{x}}{\bar{x}}\right),$$

where $\bar{x}$ is the average value of x. Show the first five rows of output. Call the percentage difference column `perdiff`. Round both `gini` and `perdiff` to three decimal places. Note that you'll have to perform a subquery here: you are selecting `gini` from `galaxies` and the average value of `gini` from a new virtual table that you have to define on the fly.

```
postgres=# select round(gini,3) as gini, round((gini/(select avg(gini) from galaxies) - 1)*100,3) as pe
 gini  | perdiff
-------+---------
 0.505 |  10.695
 0.433 |  -5.087
 0.288 | -36.871
 0.517 |  13.326
 0.303 | -33.583
(5 rows)
```

Question 9

(8 points)

Output the field name and the median value of the concentration statistic for each galaxy field, but only output those rows where that median value is greater than 3.

EDIT: the solution for this question utilizes `group_by` and `having`, two concepts covered in the notes for *Week 15*. Because students have already turned in HW 14, this question will remain; please refer to Notes 15A for more details on these two `select` clauses.

```
postgres=# select field, percentile_cont(0.5) within group (order by conc) as conc_median
from galaxies
```

```
group by field
having percentile_cont(0.5) within group (order by conc) > 3;
 field  | conc_median
--------+-------------
 COSMOS |       3.108
 GOODSN |       3.164
 GOODSS |        3.05
 UDS    |       3.122
(4 rows)
```

Question 10

(8 points)

Output the top two galaxies, ranked by Gini coefficient (decreasing), for each field. To be clear, only one table is to be output, with four columns: `field`, `gini`, `conc`, and `rank`, and ten rows (two from each field).

EDIT: when ranking, you need to utilize two commands: `partition by` and `order by`. (You "partition by" one variable and "order by" another when you `rank() over (...)`. The former is only covered in Notes 15A. Similar to Q9, this question will remain here since students have already turned in HW 14.

```
postgres=# select * from (select field, gini, conc, rank() over (partition by field order by gini desc)
 field  | gini  | conc  | rank
--------+-------+-------+------
 COSMOS | 0.948 | 0.101 |    1
 COSMOS | 0.914 | 2.878 |    2
 EGS    | 0.858 | 2.675 |    1
 EGS    | 0.840 | 2.147 |    2
 GOODSN | 0.856 | 0.585 |    1
 GOODSN | 0.830 | 3.816 |    2
 GOODSS | 0.611 | 3.508 |    1
 GOODSS | 0.587 | 3.412 |    2
 UDS    | 0.829 | 2.611 |    1
 UDS    | 0.712 | 3.483 |    2
(10 rows)
```

Download the files `cosmos_properties.csv` and `cosmos_sample.csv` from the `DATA` directory on `Canvas`. The first contains four columns: galaxy ID, age, star-formation rate, and metallicity (relative amount of heavy elements). The second contains seven columns: galaxy ID (same number as for the first file), magnitude (i.e., brightness), mass in log-base-10 solar masses, photometric redshift (coarse estimate of galaxy distance), spectroscopic redshift (precise estimate, when available; missing data are denoted `-99`), and RA and dec (celestial longitude and latitude in degrees).

Read these data files in using the following code:

```
create table properties (
  id serial primary key,
  age real,
  sfr real,
  metallicity real
);

\copy properties from 'cosmos_properties.csv' with ( format csv, header )

create table sample (
  id serial primary key,
```

```
  mag real,
  mass real,
  zbest real,
  zspec real,
  ra real,
  dec real
);

\copy sample from 'cosmos_sample.csv' with ( format csv, header )
```

Question 11

(8 points)

Regress log-base-10 of the star-formation rate upon the mass; show the slope and the intercept. (Three decimal places for each.) Only include rows for which the star-formation rate is greater than zero.

EDIT: it might appear to some that a `join` is needed here (which is something covered in Week 15), but a properly constructed `where` clause is sufficient. The same is true in Q12.

```
postgres=# select round(regr_slope(log(sfr), mass)::numeric,3) as slope, round(regr_intercept(log(sfr),
 slope  | intercept
--------+-----------
 -0.423 |     4.913
(1 row)
```

Question 12

(8 points)

Output the residual sum of squares for the linear regression model of Q11, keeping in mind that only rows for which the star-formation rate is greater than zero were included when learning the model. It may be helpful to note that the residual sum of squares is related to three quantities, S_{xx}, S_{xy}, and S_{yy}, each of which can be computed via specific functions in `postgres`. (You can use Google to determine the relationship and look into `postgres` documentation to find the functions.) Name the output column `rss` and, as usual, output the quantity to three decimal places.

```
postgres=# select round((regr_syy(log(sfr), mass) - (regr_sxy(log(sfr), mass)^2)/regr_sxx(log(sfr), mass
   rss
----------
 8084.151
(1 row)
```