OA_BigFishGames

June 8, 2022

1 Gams DS homework test

2 Applicant: Yu-Ting Shen

3 Question 1

A gamming company stores its customers, transactions, and campaigns data in 3 tales with the following format.

- customers table

id	Name	RegDate
1	Xin	2018-01-01
...	...	...

- transactions table

id	customerid	value	timestamp
1	2	0.71	2018-03-04 12:45
...	...	...	...

- campaigns table

id	customerid	Campdate
1	5	2019-01-01
...	...	...

Write SQL queries which retrieve following data 1. All customers with no transactions 2. For each customer output **mean** and **maximum** number of days between two consecutive transactions. Return "-1" for customers with no or only one transaction; e.g. if a player had transactions on "2019-01-01", "2019-01-02" and "2019-01-05" then number of days between transactions is 1 and 3, average being (1+3)/2 = 2 and maximum being 3.

3.1 *Solution*

1. SQL query for all customers with no transactions

```sql
SELECT c.id, c.Name
FROM customers AS c
LEFT JOIN transaction AS t
on c.id = t.customerid
WHERE t.value IS Null
```

2. This SQL query can be splitted into several smaller pieces.

- Firstly, we have to apply **self join** to get the day difference between two transactions. The results are save as a temporary table `t1` using `WITH` clause.
  - I use `DATE_DIFF(date_expression_a, date_expression_b, date_part)` provided by Google BigQuery. This function is equivalent to `DATEDIFF(date_part, date_expression_a, date_expression_b)` in the SQL server.
- Then we use the `t1` table to calculate the mean and maximum number of days between two consecutive transactions. The results are save as a temporary table `t2` using `WITH` caluse.
- Because there are null in `t2` table, we use the `CASE` caluse to specify the null value as "-1"

```sql
WITH t1 AS (
    SELECT tleft.customerid AS customerid,
           tleft.timestamp AS day1,
           MIN(tright.timestamp) AS day2,
           DATE_DIFF(MIN(tright.timestamp), tleft.timestamp, day) AS daydiff
    FROM transactions AS tleft
    LEFT JOIN transactions AS tright
    ON tleft.customerid = tright.customerid AND
       tleft.timestamp < tright.timestamp
    GROUP BY 1, 2
    ORDER BY 1, 2
),
t2 AS (
    SELECT customerid,
           AVG(daydiff) AS mean,
           MAX(daydiff) AS maximum
    FROM t1
    GROUP BY 1
    ORDER BY 1
)
SELECT customerid,
       CASE
           WHEN mean IS NULL THEN -1
           ELSE mean
       END AS mean,
       CASE
           WHEN maximum IS NULL THEN -1
           ELSE maximum
       END AS maximum
FROM t2
```

---

4 Question 2

A gaming campany is hiring a data scientist. Applicants are asked to participate in two online test (a) SQL and (B) coding. For each of the tests, applicants are given a score between 0 and 100. Applicants with a total score ("SQL" + "coding") of more than 100 are invited for an interview

For the question, assume that applicants' scores in these two online tests are independently uniformly distributed random variables between 0 and 100. Explain and quantify your answers to the following questions: 1. What is the correlation between SQL and coding scores of all applicants? 2. What is the correlation between SQL and coding scores of applicants who passed the test?

4.1 *Solution*

Let x_i be score of SQL and y_i be score of coding. Since x_i and y_i are independently uniformly distributed random variable between 0 and 100, we have $E(x) = E(y) = 50$. We also know the correlation coefficient is

$corr(x, y) = \frac{Cov(x,y)}{\sigma_x \sigma_y}$

1. There is **no correlation** between SQL and coding scores of all applicants because the two population X and Y are independent resulting in $Cov(x, y) = 0$.

For example, if there are 1 million applicants, we use python to calculate the Pearson correlatin coefficient, then we get the $corr(x, y) \sim 0$

[33]:
```
import numpy as np
x = np.random.uniform(0, 100, 1000000)
y = np.random.uniform(0, 100, 1000000)

print(f"mean of x = {x.mean()}, std of x = {x.std()}")
print(f"mean of y = {y.mean()}, std of y = {y.std()}")
print(f"pearson correlation r = {np.corrcoef(x, y)}")
```

```
mean of x = 50.004037783102724, std of x = 28.862379678539735
mean of y = 49.94340763930167, std of y = 28.862986313744734
pearson correlation r = [[ 1.0000000e+00 -5.8778507e-04]
 [-5.8778507e-04  1.0000000e+00]]
```

2. There is **negative correlation** between SQL and coding scores of applicants who pass the test.

[34]:
```
import pandas as pd
df = pd.DataFrame({"SQL": x, "coding": y})
df["sum"] = df["SQL"] + df["coding"]
df_pass = df.loc[df["sum"] > 100]

print(f"mean of SQL = {df_pass['SQL'].mean()}, std of SQL = {df_pass['SQL'].
 ↪std()}")
print(f"mean of coding = {df_pass['coding'].std()}, std of coding =␣
 ↪{df_pass['coding'].std()}")
print(f"pearson correlation r = ", df_pass["SQL"].corr(df_pass["coding"]))
```

```
# print(df_pass["coding"].corr(df_pass["SQL"]))
```

```
mean of SQL = 66.70334299247783, std of SQL = 23.550025628729223
mean of coding = 23.58028168589552, std of coding = 23.58028168589552
pearson correlation r =  -0.499819851891228
```

---

5 Question 3

Order the following situations based on the expected power of the statistical test intended to identify the difference between two groups. Explain your reasoning.

1. Datasets come from normal distributions with equal unknown variances
2. Datasets come from normal distributions with unequal known variances
3. Datasets come from two unknown distributions
4. Datasets come from normal distributions with unequal known variances and known unequal means
5. Datasets come from gamma distributions with unknown parameters

5.1 *Solution*

Expected power: 4 > 2 > 1 > 5 > 3

The statistical power is the probability that the test correctly rejects the null hypothesis. Since we want to use the statistical test to identify the difference between groups, the null and alternate hypothesis are

- H_0: $\mu_1 = \mu_2$, the two groups are the same
- H_a: $\mu_1 \neq \mu_2$, the two groups are different

We either use two-sample z-test or two-sample t-test for this purpose. Based on the definition of these two tests, if the variance is known, then we use z-test. Otherwise, we use t-test.

- Normal distribution + unknown variance $\rightarrow$ use t-test
- Normal distribution + known variance $\rightarrow$ use z-test

For case 4, we know $\mu_1 \neq \mu_2$ and $s_1 \neq s_2$, we can correctly reject the null hypothesis, therefore, it has the best power.

For case 2, we know the variances of two groups, then z-test is used. For case 1, we don't know the variances of two groups, then t-test is used. Usually, the z-test has better power to reject H_0 correctly than the t-test.

If we know the distribution is gamma distribution but we don't know the parameters (case 5), then the power to reject H_0 correctly definitely better than we don't know the distribution (case 3).

---

6 Question 4

Write a function in **R** or **Python** which takes a vector length L consisting only of 0s and 1s as input and returns the largest number of consecutive 1s from the vector as an integer, e.g. the function takes [1,1,0,1,1,1,0,0,1] and returns 3

6.1 *Solution*

We can use `longest` to keep the longest consecutive 1s and use `temp` to keep the current consecutive 1s. When we loop over all elements in the input vector, we add `1` into `temp` if the element is one and we reset `temp` to `0` if the element is zero. And we compare the `temp` with `longest`, if the `temp` is larger than the `longest`, then we replace the value of `longest` by the value of `temp`. At the end of the `for` loop we can get the largest number of consecutive 1s. * Time complexity = $O(n)$ * Space complexity = $O(1)$

[2]:
```python
def longest_consecutive(nums):
    if nums is None or len(nums) == 0:
        return 0

    longest = 0 # keep the longest consecutive 1s
    temp = 0 # current consecutive
    for i in range(len(nums)):
        if nums[i] == 1:
            temp += nums[i]
        else:
            temp = 0
        if temp > longest:
            longest = temp
    return longest
```

[5]:
```python
# test case:
print(longest_consecutive([1,1,0,1,1,1,0,0,1]))
print(longest_consecutive([]))
print(longest_consecutive([0,0,0,0,0,0,1]))
print(longest_consecutive([1,1,1,1]))
print(longest_consecutive([0,1,0,1,0,1]))
nums = [0,1,1,1,1,1,0,1,1,1,
        0,0,0,0,1,1,1,1,0,0,
        0,0,0,1,1,1,1,1,1,1,
        1,0,0,0,0,0,0,0,1,1,
        1,1,0,1,1,0]
print(longest_consecutive(nums))
```

```
3
0
1
4
1
```

```
8
```

```
[ ]:
```