Inkitt - DS/Analyst Tech Challenge

Instructions

Please work on the below three tasks and send them back to us **48 hours** after you received this challenge. The first two tasks are SQL-focused, while the third one is python-focused.

Please submit all code in a jupyter notebook (.ipynb) and attach a PDF version of it as well.

We will review your submission and assess effectiveness, efficiency, and readability. You won't fail automatically when you make a small mistake. Please make sure that your code is well-commented, well-formatted and easy to read.

Likewise, if you don't manage to complete a particular task, that's no reason for immediately failing the challenge. Some tasks are designed to be harder than others. We do expect candidates for senior positions to be able to complete the challenge.

Task 2b is a bonus question. This means that if you don't complete it, it won't reflect badly on your submission. However, we do expect candidates for a senior position to be able to complete this.

We recommend that if you feel you may run out of time before completing all tasks choose strategically. Rather than "roughly" completing all tasks, only focus on some and work out a good solution. Rather skip one of the SQL tasks than skip the Python task.

Some tasks may require you to make specific assumptions. Please document these assumptions explicitly in your submission.

If you pass this step in the hiring process, please be ready to discuss your submission in a later interview step.

Good luck! Hopefully this challenge gives a glimpse into the type of problems we work on at Inkitt.

SQL

At Inkitt we primarily work with Postgres and Redshift databases. If you are not familiar with the SQL dialects of these DBs, please use one that you are familiar with. In any case, please clearly indicate which dialect you use.

The SQL tasks **do not relate to the CSV** data **-** the data is used in the Python task. For the SQL tasks we will simply provide table descriptions and data samples in the text.

To submit your SQL code via the jupyter notebook, it's sufficient to just add it as a string definition like this:

```
task_1_query = '''
YOUR SQL QUERY
'''
```

Task 1: Story Performance

A story in our app Galatea consists of chapters. A user reads a chapter (event "read" is generated) and then hits the paywall to continue to the next chapter where she can either do a purchase (event "purchase" is generated) or wait for 6h to read the next chapter.

Event names can have following values: "read", "purchase".

We have the following table:

```
Table Name: events
Columns:
- user_id (int)
- event_name (string)
- story (string)
- chapter (int)
- timestamp (datetime)
```

Sample data from the table:

```
| user_id    | event_name  | story  | chapter | timestamp
------------------------------------------------------------------
|          1 |   read      | story1 |       1 | 2020-08-01 18:05:56
|          1 |   read      | story1 |       2 | 2020-08-01 18:12:01
|          1 |   purchase  | story1 |       2 | 2020-08-01 18:13:33
|          1 |   read      | story1 |       3 | 2020-08-01 18:31:49
|          1 |   read      | story1 |       4 | 2020-08-02 09:09:22
|          1 |   purchase  | story1 |       5 | 2020-08-01 09:20:11
|          1 |   read      | story2 |       1 | 2020-08-01 09:31:47
|          2 |   read      | story3 |       1 | 2020-09-12 23:09:16
|          2 |   purchase  | story3 |       1 | 2020-09-12 23:11:32
|          2 |   read      | story3 |       2 | 2020-09-12 23:29:58
|          3 |   read      | story3 |       1 | 2020-08-01 18:05:56
```

a) Calculate average purchases per reader per story for September 2021.

b) Rank (use dense ranking) stories based on this with the highest purchases per reader average as 1. You can submit a) and b) in the same query.

Desired output:

```
story     | purchases_per_reader  | rank
________________________________________
story1    |                  2    |   78
story2    |                  0    |  435
story3    |                0.5    |  305
```

Task 2: Subscriptions

Suppose users can also buy subscriptions. So, instead of unlocking individual chapters, users can purchase a subscription and unlock all content while the subscription is running.

We collect data on subscriptions with three events:

- "purchase": a user purchases a subscription (we offer 1-month and 12-month subscriptions).
- "renewal": an existing subscription renews automatically (exactly after the subscription period ends)
- "cancel": a user cancels her subscription (the subscription will continue running up to the end of the current subscription period, but won't renew)

The event data is collected in the following table:

```
Table Name: events
Columns:
- user_id (int)
- event_name (string)
- subscription_name (string, the type of subscription purchased)
- price (float)
- subscription_id (int, identifier of the individual subscription)
- timestamp (datetime)
```

Notes:

- Price is in € after conversion from local currencies. You can assume that we charge the same amount at initial purchase and subsequent renewals.
- Only the purchase events contain information on subscription name and price.

Sample data from the table:

```
| user_id    | event_name  | subscription_name | price | subscription_id | timestamp
-------------------------------------------------------------------------------------------
|       1    |   purchase  | subs-1month       |  7.99 |             863 | 2020-08-01 18:05:56
|       1    |   renewal   |                   |       |             863 | 2020-09-01 21:12:01
|       1    |   renewal   |                   |       |             863 | 2020-10-01 17:13:33
|       1    |   renewal   |                   |       |             863 | 2020-11-01 22:31:49
|       1    |   cancel    |                   |       |             863 | 2020-11-02 09:10:22
|       1    |   purchase  | subs-12month      | 64.99 |            1244 | 2020-11-23 15:30:11
|       1    |   renewal   |                   |       |            1244 | 2021-11-23 13:31:47
|       2    |   purchase  | subs-12month      | 61.87 |            1076 | 2020-09-15 23:09:16
|       2    |   cancel    |                   |       |            1076 | 2020-10-14 08:21:32
|       3    |   purchase  | subs-1month       |  8.31 |            1384 | 2021-01-09 18:58:01
|       3    |   renewal   |                   |       |            1384 | 2021-02-09 16:11:48
|       4    |   purchase  | subs-12month      | 64.99 |            1771 | 2021-04-15 05:51:49
```

a) **Calculate the total revenue per month from subscriptions, once including renewal revenues, and once excluding renewal revenues**

Desired output:
```
| month   | subscription_revenue  | subscription_revenue_excl_renewal
----------------------------------------------------------------------
| 2020-01 |          113439.55  |                          50821.81
| 2020-02 |          124711.08  |                          52349.35
| 2020-03 |          151932.39  |                          68236.01
```

b) **[BONUS] Write a query that produces the subscription status for each subscription (can be 'active', 'active_cancelled', 'ended') as well as start and end date.**

The end date for ended and cancelled subscriptions is the date at which the subscription runs out. The end date for active subscriptions is the date before the next expected renewal.

Desired output:
```
| subscription_id | start_date  | end_date    | status
-------------------------------------------------------------------
|             863 | 2020-08-01  | 2020-11-30  | ended
|            1673 | 2021-06-26  | 2022-06-25  | active_cancelled
|            1673 | 2021-12-28  | 2022-12-27  | active
```

Part 2: Python

Please prepare all your code in the same jupyter notebook as the SQL part and in a way that it can easily be re-run.

Please make use of proven and widely used packages instead of obscure ones whenever possible.

Task 3: Story performance

The attached CSV file contains real, but sampled, user reading data from our Galatea app. It tracks per user which story and chapter was started (event "chapter_start") and read (event "chapter_read").

We have the following table:

```
Filename: story_data.csv
Columns:
- user_id
- event_name
- story_id
- chapter
- timestamp
```

a) **Calculate chapter retention rates (for a given story chapter: *(# readers of chapter / (# starters of the story)*) and visualise them in a way that allows to compare them across stories. Based on these rates, which story performs best?**

b) **Which other metrics measuring story performance can you think of? Feel free to calculate those that can be constructed based on the above data. For those you are missing data for, please explain what kind of data you'd like to use. Please briefly describe pros and cons of the metrics you identified, especially in relation to chapter retention rates from a).**