# Social Computing Homework Coursebook

**Instructions**
Please fill in each exercise and submit the entire document as a PDF on Moodle before the section's respective deadline. Keep working on the whole document so that for the last submission you submit a completely filled in template. You may not change previous sections in subsequent submissions. Some sections require you to work on an existing software project, which you have to fork on GitHub.com, or clone and create your repository. Provide the URL of your public fork or repository of this project below.

Fill in each answer to a homework task to the textbox underneath. Use as much space as you wish. Do not provide long code snippets or other irrelevant information.

**Restrictions**
You may use AI tools for language styling or only. Usage of any AI tools to answer questions, inspire creative solutions or write code is strictly forbidden. Group work and sharing solutions is strictly prohibited. Any suspected cases of misconduct will be referred to the Education Dean. If you are not sure whether you are in violation of course-specific restrictions or the university's code of conduct, please ask the Lecturer or a TA.

**Name**

*Vu Truong*

**Student ID**

*2506393*

**Student Email**

*Vu.Truong@student.oulu.fi*

**GitHub Repository URL**

*https://github.com/HarryxDD/social-computing-hw*

# AI Use Disclaimer

**Explain in detail in what parts and how AI was used for any of the work above. Fill it out and update after each homework submission, even if you did not use AI at all.**

**Your answers to homework tasks should not include AI-generated code or text.**

*I used Social Computing AI Agent to ask about the task 1.1 because there are no "purpose" description inside the database.*

*For task 2.1, I used AI Agent to ask about the growing trend of some social media platforms.*

## Task 1 (due 22.9.2025 23:59) 15 points

**Exercise 1.1** Reading the dataset: Load the database and for each table, print and inspect the available columns and the number of rows. Explain below how you loaded the database. For each table, describe all columns (name, purpose, type, example of contents). You may use SQL and/or Python to perform this task. (3 points)

```
# Explanations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location (same location as the code file)
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

# Read all table names -> turn it to a dataframe
tablenames_df = pd.read_sql_query("SELECT name FROM sqlite_master WHERE
type='table';", conn)

# Convert df to a list
tables = tablenames_df['name'].tolist()

for table in tables:
    print(f"Table: {table}")
    df = pd.read_sql_query(f"SELECT * FROM {table};", conn)
    # Inspect the table
    print(f"Number of rows: {len(df)}")
    print(f"Available columns: {df.columns.tolist()}")
    # Get metadata
    col = pd.read_sql_query(f"PRAGMA table_info({table});", conn)
    for idx, row in col.iterrows():
        print(f"Name: {row['name']}")
        print(f"Type: {row['type']}")
        # Hardcoded purpose as metadata is not available in db so I will
describe this in the output
        print(f"Purpose: -")
        print(f"Example: {df[row['name']].head(1).values[0]}")
        print("--")
    print("-----")

"""
Output:
```

```
Table: follows
Note: This is a many-to-many relationship table between users and their
followers
Number of rows: 7225
Available columns: ['follower_id', 'followed_id']
Name: follower_id
Type: INT
Purpose: This is the id of the user who is following
Example: 12
--
Name: followed_id
Type: INT
Purpose: This is the id of the user who is being followed
Example: 1
--
-----
Table: users
Number of rows: 210
Available columns: ['id', 'username', 'location', 'birthdate', 'created_at',
'profile', 'password']
Name: id
Type: INT
Purpose: Id of the user
Example: 1
--
Name: username
Type: varchar(50)
Purpose: Username of user
Example: artistic_amy
--
Name: location
Type: varchar(100)
Purpose: Location of user
Example: Boston, USA
--
Name: birthdate
Type: date
Purpose: User's date of birth
Example: 1997-06-30
--
Name: created_at
Type: timestamp
Purpose: The timestamp when the user account was created
```

```
Example: 2022-07-01 12:17:48
--
Name: profile
Type: TEXT
Purpose: Profile description of user that contains personality traits and
interests
Example: Artistic soul from Boston ? | Born in '97 | Balancing mind & style |
Fashion lover | News junkie | Embracing the highs and lows | Dreaming big,
moving forward ✨
--
Name: password
Type: TEXT
Purpose: Password for the account
Example: izmQoLHw
--
-----
Table: sqlite_sequence
Note: Automatically created table manage AUTOINCREMENT fields
Number of rows: 3
Available columns: ['name', 'seq']
Name: name
Type:
Purpose: Shows which table (like reactions, posts, ect) the row is about
Example: reactions
--
Name: seq
Type:
Purpose: Shows the last used AUTOINCREMENT value for that table
Example: 8286
--
-----
Table: reactions
Number of rows: 8276
Available columns: ['id', 'post_id', 'user_id', 'reaction_type']
Name: id
Type: INTEGER
Purpose: Id of the reaction
Example: 1
--
Name: post_id
Type: INTEGER
Purpose: Id of the post that the reaction is for
Example: 2631
--
```

```
Name: user_id
Type: INTEGER
Purpose: Id of the user who made the reaction
Example: 60
--
Name: reaction_type
Type: TEXT
Purpose: The type of reaction
Example: like
--
-----
Table: comments
Number of rows: 5804
Available columns: ['id', 'post_id', 'user_id', 'content', 'created_at']
Name: id
Type: INTEGER
Purpose: Id of the comment
Example: 1
--
Name: post_id
Type: INTEGER
Purpose: Id of the post that the comment is for
Example: 1963
--
Name: user_id
Type: INTEGER
Purpose: Id of the user who commented
Example: 55
--
Name: content
Type: TEXT
Purpose: Content of the comment
Example: Haha, I bet your neighbors are either loving or hating you right now!
Crank it up and see if you can get a dance party going next door. #DIYparty
--
Name: created_at
Type: TIMESTAMP
Purpose: The timestamp when the comment was created
Example: 2022-12-04 02:36:15
--
-----
Table: posts
Number of rows: 1303
Available columns: ['id', 'user_id', 'content', 'created_at']
```

```
Name: id
Type: INTEGER
Purpose: Id of the post
Example: 1718
--
Name: user_id
Type: INTEGER
Purpose: Id of the post owner
Example: 10
--
Name: content
Type: TEXT
Purpose: Content of the post
Example: Just had the most ridiculous encounter with a cat in Shibuya. It
hissed like I was invading its turf! #CatWhisperer #TokyoLife
--
Name: created_at
Type: TIMESTAMP
Purpose: The timestamp when the post was created
Example: 2023-10-12 10:43:24
--
-----

"""
```

**Exercise 1.2** Lurkers: How many users are there on the platform who have not interacted with posts or posted any content yet (but may have followed other users)? Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (3 points)

```python
# Explanations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

try:
    # Check for users who not exist in posts, comments, and reactions table
using subqueries
```

```
    lurkers = pd.read_sql_query("""
    SELECT
        id
    FROM users
    WHERE id NOT IN (SELECT user_id FROM posts)
    AND id NOT IN (SELECT user_id FROM comments)
    AND id NOT IN (SELECT user_id FROM reactions);
    """, conn)
    # print("Lurkers: ")
    # print(lurkers)
    print("The number of people who have not interacted at all: ",
len(lurkers))
except Exception as e:
    print(f"Error: {e}")

"""
Output:
The number of people who have not interacted at all:  55

"""
```

**Exercise 1.3** Influencers: In the history of the platform, who are the 5 users with the most engagement on their posts? Describe how you measure engagement. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (4 points)

```
# Explanations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

"""
To find top 5 influencers, I count the number of reactions and comments on each
user's posts.
First, I JOIN the posts table with the users table to get the username (the
author).
Then, I LEFT JOIN the reactions and comments tables to count the number of
reactions and comments for each posts.
```

```
Finally, I group the results by username and order them by the total number of
reactions and comments in descending order, limiting the results to the top 5.

By using DISTINCT in the COUNT, I ensure that each reaction and is counted only
once, because when joining multiple tables, there can be duplicate rows for the
same reaction and comment, resulting in same count value for these columns.
"""

try:
    influencer_df = pd.read_sql_query("""
    SELECT
        users.id,
        users.username,
        COUNT(DISTINCT reactions.id) as Reactions,
        COUNT(DISTINCT comments.id) AS Comments
    FROM posts
    JOIN users on users.id = posts.user_id
    LEFT JOIN reactions on posts.id = reactions.post_id
    LEFT JOIN comments ON posts.id = comments.post_id
    GROUP by users.username
    ORDER BY (COUNT(DISTINCT reactions.id) + COUNT(DISTINCT comments.id)) DESC
    LIMIT 5;
    """, conn)
    print("Top 5 influencers: ")
    print(influencer_df)
except Exception as e:
    print(f"Error: {e}")

"""
Output:
Top 5 influencers:
   id      username  Reactions  Comments
0  54    WinterWolf        267       179
1  65   PinkPanther        234       152
2  94     PinkPetal        246       137
3  81  GoldenDreams        217       149
4  30     WildHorse        196       157

"""
```

**Exercise 1.4** Spammers: Identify users who have shared the same text in posts or comments at least 3 times over and over again (in all their history, not just the last 3 contributions). Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (5 points)

```python
# Explanations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

"""
For this task, I identify spammer by check the same contents being posted or
commented more than 3 times by the same user
I use 2 separate SELECT to find the spam and combine them using UNION.
I also add a column 'type' to indicate whether the spam is from post or
comment.
"""

try:
    spammer_df = pd.read_sql_query("""
    SELECT
        users.username,
        posts.content,
        'post' as type,
        COUNT(*) as occur
    FROM posts
    JOIN users on users.id = posts.user_id
    GROUP by posts.user_id, posts.content
    HAVING COUNT(*) >= 3

    UNION

    SELECT
        users.username,
        comments.content,
        'comment' as type,
        COUNT(*) as occur
    FROM comments
    JOIN users on users.id = comments.user_id
    GROUP by comments.user_id, comments.content
    HAVING COUNT(*) >= 3;
    """, conn)
    print("Spammer: ")
    print(spammer_df)
```

```
except Exception as e:
    print(f"Error: {e}")

"""
Output:
Spammer:
        username                                              content     type  o
ccur
0    coding_whiz  ?FREE VACATION? Tag a friend you'd take to
Bal...  comment      3
1    coding_whiz  Shocking! #lol #weekend #coffee #bookstagram
#...     post      3
2    coding_whiz  Top 10 gadgets of 2025 – All available here:
b...     post      8
3    eco_warrior  Not gonna lie, I was skeptical at first. But
a...     post      6
4    eco_warrior  Revolutionary idea! #fashionblogger
#instafash...     post      3
5    eco_warrior  Wearing this hoodie in my latest reel—so many
...     post      4
6   history_buff  A lot of you asked what helped me drop 5kg in
...     post      5
7   history_buff  Best way to clean your sneakers ? snag yours
h...     post      5
8   history_buff  Mood: me refreshing for likes every 30
seconds...     post      5
9   history_buff  What do you think? #thoughts
#motivationmonday...     post      4
10  history_buff  You need this travel pillow in your life ?
sho...     post      3
11     night_owl  ? Mega Giveaway Alert! ? Follow all accounts
w...     post      8
12     night_owl  ?FLASH GIVEAWAY? Click the link in our bio to
...     post      5
13     night_owl  Find out why everyone is switching to this
new...     post      4
14     night_owl  This one trick will make you $500/day from
hom...     post      3
15     yoga_yogi  I couldn't believe it! I just entered this
giv...     post      5
16     yoga_yogi  Just entered this Xbox giveaway and the form
w...     post      3

"""
```

## Task 2 (due 29.9.2025 23:59) 15 points

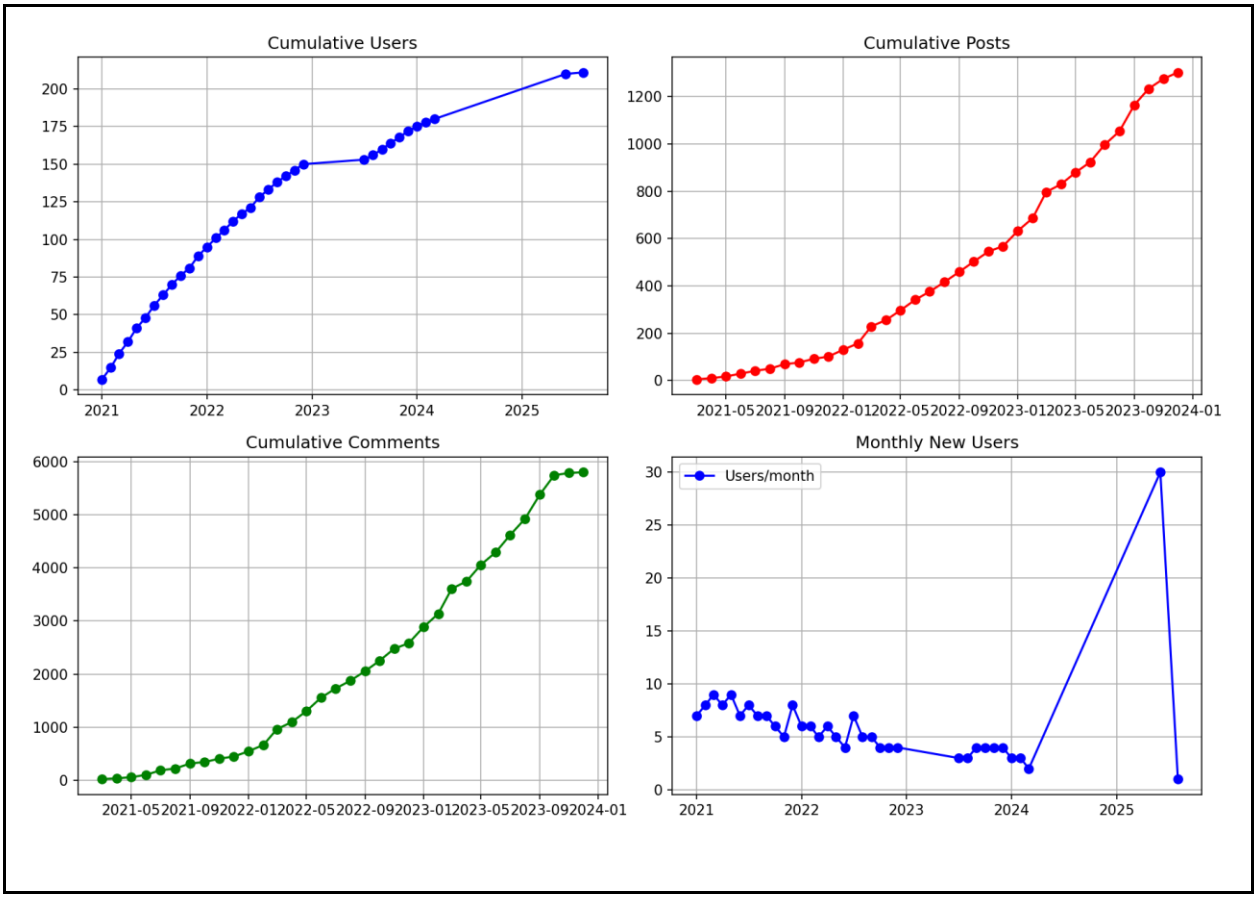
**Exercise 2.1** Growth: This year, we are renting 16 servers to run our social media platform. They are soon at 100% capacity, so we need to rent more servers. We would like to rent enough to last for 3 more years without upgrades, plus 20% capacity for redundancy. We need an estimate of how many servers we need to start renting based on past growth trends. Plot the trend on a graph using Python and include it below. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (Note that the dataset may not end in the current year, please assume that the last data marks today's date) (3 points)

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

"""
For this task, I thought about the growing factors, is it linear growth or
exponential growth, as normally some social media platforms grow exponentially
in the beginning, but after a while, the growth rate slows down.
After analyzing the data, I found that the growth is more linear than
exponential. So I decided to use a linear projection for the next 3 years.

And the answer for the number of additional servers needed is 23. The
calculation will be shown below.

"""

def get_data():
    conn = sqlite3.connect('database.sqlite')

    # This query get total counts of users, posts, and comments.
    totals = pd.read_sql_query("SELECT (SELECT COUNT(*) FROM users) as users,
(SELECT COUNT(*) FROM posts) as posts, (SELECT COUNT(*) FROM comments) as
comments", conn)

    # These queries get monthly new users, posts, and comments.
```

```
    monthly_users = pd.read_sql_query("SELECT strftime('%Y-%m', created_at) as
month, COUNT(*) as count FROM users GROUP BY strftime('%Y-%m', created_at)
ORDER BY month", conn)
    monthly_posts = pd.read_sql_query("SELECT strftime('%Y-%m', created_at) as
month, COUNT(*) as count FROM posts GROUP BY strftime('%Y-%m', created_at)
ORDER BY month", conn)
    monthly_comments = pd.read_sql_query("SELECT strftime('%Y-%m', created_at)
as month, COUNT(*) as count FROM comments GROUP BY strftime('%Y-%m',
created_at) ORDER BY month", conn)

    conn.close()

    return totals.iloc[0]['users'], totals.iloc[0]['posts'],
totals.iloc[0]['comments'], monthly_users, monthly_posts, monthly_comments

def calculate_projections(total_users, total_posts, total_comments,
monthly_users):
    # The value 1.0 is based on the assumption that each user has many props,
such as posts, comments, authentication, etc.
    user_weight = 1.0

    # For posts, each of them can contains long text, images, and interactions.
    post_weight = 0.5

    # For comments, they are usually short text, but can also contain images,
and reactions.
    comment_weight = 0.2

    # Traffic spike factor to account for peak times when user activity is
higher.
    traffic_spike_factor = 1.2

    # Current server load
    current_load = (total_users * user_weight + total_posts * post_weight +
total_comments * comment_weight) * traffic_spike_factor

    # Continue current growth for 3 years
    days_until_now = len(monthly_users) * 30
    daily_user_growth = total_users / days_until_now

    # Projected number of users for the next 3 years
    projected_users = total_users + (daily_user_growth * 1095)

    # Projected posts and comments based on user growth
```

```
    user_growth_multiplier = projected_users / total_users
    projected_posts = total_posts * user_growth_multiplier
    projected_comments = total_comments * user_growth_multiplier

    # Calculate projected server load and servers needed
    projected_load = (projected_users * user_weight + projected_posts *
post_weight + projected_comments * comment_weight) * traffic_spike_factor

    # Current servers with 20% redundancy
    needed_servers = 16 * (projected_load / current_load) * 1.2

    return {
        'users': projected_users,
        'posts': projected_posts,
        'comments': projected_comments,
        'needed_servers': needed_servers
    }

def create_plots(monthly_users, monthly_posts, monthly_comments):
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 8))

    for df in [monthly_users, monthly_posts, monthly_comments]:
        df['date'] = pd.to_datetime(df['month'])
        df['cumulative'] = df['count'].cumsum()

    # Create 4 plots
    ax1.plot(monthly_users['date'], monthly_users['cumulative'], 'b-o')
    ax1.set_title('Cumulative Users'); ax1.grid(True)

    ax2.plot(monthly_posts['date'], monthly_posts['cumulative'], 'r-o')
    ax2.set_title('Cumulative Posts'); ax2.grid(True)

    ax3.plot(monthly_comments['date'], monthly_comments['cumulative'], 'g-o')
    ax3.set_title('Cumulative Comments'); ax3.grid(True)

    ax4.plot(monthly_users['date'], monthly_users['count'], 'b-o',
label='Users/month')
    ax4.set_title('Monthly New Users'); ax4.grid(True); ax4.legend()

    plt.tight_layout()
    plt.savefig('growth_analysis.png', dpi=150)
    plt.show()

def analyze_and_plot():
```

```
    total_users, total_posts, total_comments, monthly_users, monthly_posts,
monthly_comments = get_data()

    print(f"Current: {total_users} users, {total_posts} posts, {total_comments}
comments")

    results = calculate_projections(total_users, total_posts, total_comments,
monthly_users)

    print(f"\n3-Year Linear Projection:")
    print(f"  Users: {results['users']:.0f}, Posts: {results['posts']:.0f},
Comments: {results['comments']:.0f}")
    print(f"  Additional servers needed: +{results['needed_servers'] -
16:.0f}")
    print(f"  Total servers: {results['needed_servers']:.0f}")

    create_plots(monthly_users, monthly_posts, monthly_comments)

if __name__ == "__main__":
    analyze_and_plot()

"""
Output:
Current: 211 users, 1303 posts, 5804 comments

3-Year Linear Projection:
  Users: 431, Posts: 2662, Comments: 11857
  Additional servers needed: +23
  Total servers: 39

"""
```




**Exercise 2.2** Virality: Identify the 3 most viral posts in the history of the platform. Select and justify a specific metric or requirements for a post to be considered viral. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (4 points)

```
# Explainations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

"""
After research about viral post, I found that it is a piece of content that
gets shared quickly across various social media platforms in a short period of
time. So I decided to use growth rate in the first few hours to measure the
virality of a post.
```

```
I was trying to calculate the growth rate based on reactions, but I found that
the table does not have a created_at column, so I can only use comments in this
case.
"""

CALCULATING_HOURS = 24

def calculate_growth_rate_hours(table_alias, post_alias, hours):
    # Don't forget to check if the hours since posted is less than the
calculating hours
    return f"""
    COUNT(DISTINCT CASE WHEN (julianday({table_alias}.created_at) -
julianday({post_alias}.created_at)) * 24 <= {hours} THEN {table_alias}.id END)
* 1.0 /
    CASE
        WHEN (julianday('now') - julianday({post_alias}.created_at)) * 24 >=
{hours} THEN {hours}
        WHEN (julianday('now') - julianday({post_alias}.created_at)) * 24 < 1
THEN 1
        ELSE (julianday('now') - julianday({post_alias}.created_at)) * 24
    END
    """

try:
    viral_post_df = pd.read_sql_query(f"""
    SELECT
        p.id,

        -- Total engagement (comments + reactions)
        COUNT(DISTINCT c.id) as total_comments,
        COUNT(DISTINCT r.id) as total_reactions,
        (COUNT(DISTINCT c.id) + COUNT(DISTINCT r.id)) as absolute_engagement,

        -- Growth rate: comments per hour in first {CALCULATING_HOURS} hours
        {calculate_growth_rate_hours('c', 'p', CALCULATING_HOURS)} as
growth_rate,

        -- Combined virality score
        {calculate_growth_rate_hours('c', 'p', CALCULATING_HOURS)} *
(COUNT(DISTINCT c.id) + COUNT(DISTINCT r.id)) as virality_score
    FROM posts p
    LEFT JOIN comments c on c.post_id = p.id
    LEFT JOIN reactions r on r.post_id = p.id
    GROUP by p.id
```

```
    HAVING absolute_engagement > 0
    ORDER BY virality_score DESC
    LIMIT 3;
    """, conn)
    print(f"Viral posts - first {CALCULATING_HOURS} hours: ")
    print(viral_post_df)
except Exception as e:
    print(f"Error: {e}")

"""
Output:

Viral posts - first 5 hours:
     id  total_comments  total_reactions  absolute_engagement  growth_rate  vir
ality_score
0  2351              62              139                  201         12.4
     2492.4
1  2813              82              103                  185         12.0
     2220.0
2  2195              45              133                  178          9.0
     1602.0

Viral posts - first 12 hours:
     id  total_comments  total_reactions  absolute_engagement  growth_rate  vir
ality_score
0  2813              82              103                  185     6.833333
1264.166667
1  2351              62              139                  201     5.166667
1038.500000
2  2004              71               94                  165     5.916667
 976.250000

Viral posts - first 24 hours:
     id  total_comments  total_reactions  absolute_engagement  growth_rate  vir
ality_score
0  2813              82              103                  185     3.416667
 632.083333
1  2351              62              139                  201     2.583333
 519.250000
2  2004              71               94                  165     2.958333
 488.125000

As we can see, the vital posts are consistent across different hours, so the
answer for the question is post id 2813, 2351, and 2004. There was a slight
```

```
change in the order because there's a higher early burst of post id 2351 at the
start, but slower sustained growth.
"""
```

**Exercise 2.3** Content Lifecycle: What is the average time between the publishing of a post and the first engagement it receives? What is the average time between the publishing of a post and the last engagement it receives? Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (4 points)

```
# Explainations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
conn = sqlite3.connect(dbfile)

"""
For this task, I define the engagement based on comments since the reactions
table does not have a created_at column.

I excluded posts that have no comments, since they do not have any engagement,
but still show the number of such posts in the output.

Basically, I created a CTE to calculate the time to first comment and time to
last comment for each post, then I used aggregate functions to get the required
metrics.

I used INNER JOIN to exclude posts with no comments first, then I calculated
the number of such posts by subtracting from the total.
"""

try:
    content_lifecycle = pd.read_sql_query(f"""
    with post_lifecycle as (
    SELECT
        p.id,
        p.created_at,
        MIN(c.created_at) AS first_comment_at,
```

```
        (julianday(MIN(c.created_at)) - julianday(p.created_at)) * 24 as
hours_to_first_comment,
        MAX(c.created_at) as last_comment_at,
        (julianday(MAX(c.created_at)) - julianday(p.created_at)) * 24 as
hours_to_last_comment
    from posts p
    INNER join comments c on p.id = c.post_id
    GROUP by p.id
    )
    SELECT
        COUNT(*) as posts_with_comments,
        (select COUNT(*) from posts) - count(*) as posts_with_no_comments,
        AVG(hours_to_first_comment) as avg_hr_to_first_cmt,
        AVG(hours_to_last_comment) as avg_hr_to_last_cmt
    from post_lifecycle;
    """, conn)
    print(f"Content Lifecycle: ")
    print(content_lifecycle)
except Exception as e:
    print(f"Error: {e}")

"""
Output:
Content Lifecycle:
   posts_with_comments  posts_with_no_comments  avg_hr_to_first_cmt  avg_hr_to_
last_cmt
0                 1215                      88            86.604362          15
1.445664

"""
```

**Exercise 2.4** Connections: Identify the top 3 user pairs who engage with each other's content the most. Define and describe your metric for engagement. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (4 points)

```
# Explainations for the work are being added as comments
import sqlite3
import pandas as pd

# Current db file location
dbfile = 'database.sqlite'
# Establish a connection to the db
```

```
conn = sqlite3.connect(dbfile)

"""
For this task, I define engagement as the total number of comments and
reactions exchanged between two users on each other's posts. This means I count
all individual comments and reactions that flow in both directions between a
user pair.

First, I create the CTE all_engagements to gather all comments and reactions
between users, ensuring that self-engagements are excluded by using WHERE
(c|r).user_id != p.user_id.

The second CTE user_pairs aggregates the total engagement between each pair of
users.
For example, if User A commented 2 times and reacted 3 times to User B's posts,
the total engagement from User A to User B would be 5.

The third CTE mutual_engagement combines the engagements from both users in
each pair to get the total mutual engagement. I joined the user_pairs table
with itself to achieve this. I avoid double counting by ensuring that I only
consider pairs where action_owner < post_owner (or action_owner > post_owner no
matter), so each user pair appears only once in the final results regardless of
who initiated more engagement.

"""

try:
    connections = pd.read_sql_query(f"""
    WITH all_engagements AS (
    SELECT
        c.user_id AS action_owner,
        p.user_id AS post_owner,
        'comment' AS type,
        count(*) AS quantity
    FROM comments c
    JOIN posts p ON p.id = c.post_id
    WHERE c.user_id != p.user_id
    GROUP BY c.user_id, p.user_id

    UNION ALL

    SELECT
        r.user_id AS action_owner,
        p.user_id AS post_owner,
```

```
        'reaction' AS type,
        count(*) AS quantity
    FROM reactions r
    JOIN posts p ON p.id = r.post_id
    WHERE r.user_id != p.user_id
    GROUP BY r.user_id, p.user_id
    ),
    user_pairs as (
    SELECT
        action_owner,
        post_owner,
        SUM(quantity) AS total_engagement
    FROM all_engagements
    GROUP BY action_owner, post_owner
    ),
    mutual_engagement AS (
    SELECT
        CASE WHEN e1.action_owner < e1.post_owner THEN e1.action_owner ELSE
e1.post_owner END AS user1_id,
        CASE WHEN e1.action_owner < e1.post_owner THEN e1.post_owner ELSE
e1.action_owner END AS user2_id,
        e1.total_engagement + e2.total_engagement AS mutual_total
    FROM user_pairs e1
    JOIN user_pairs e2 ON e1.action_owner = e2.post_owner AND e1.post_owner =
e2.action_owner
    WHERE e1.action_owner < e1.post_owner
    )

    SELECT
        u1.username AS user1,
        u2.username AS user2,
        me.mutual_total AS total_mutual_engagement
    FROM mutual_engagement me
    JOIN users u1 ON me.user1_id = u1.id
    JOIN users u2 ON me.user2_id = u2.id
    ORDER BY me.mutual_total DESC
    LIMIT 3;
    """, conn)
    print(f"Connections: ")
    print(connections)
except Exception as e:
    print(f"Error: {e}")

"""
```

```
Output:
Connections:
           user1       user2  total_mutual_engagement
0  DancingDolphin  SilverMoon                       16
1     userInBlack    TigerEye                       13
2       StarGazer  WinterWolf                       13

"""
```

## Task 3 (due 19.10.2025 23:59) 15 points

**Exercise 3.1** Censorship: implement the `moderate_content` function that automatically detects and censors inappropriate user posts on the platform. Your function should take a post, comment or user introduction as input and apply censorship rules to either clean or remove content, and supply a risk score that corresponds to the number and weight of violations in the content (note the risk classification thresholds in the code). The exact rules are detailed on the Rules page. Think of and implement one more moderation measure you think is important to keep the platform safe. Include and explain your implementation below. (5 points)

*Write your answer here…*

**Exercise 3.2** User risk analysis: Assign risk scores to each user by implementing the `user_risk_analysis` function. This function returns a risk score for a given user based on rules presented on the Rules page. Identify the top 5 highest risk users. Think of and implement one more risk prediction measure you think is important to keep the platform safe. Answer and explain your queries/calculations below. (5 points)

*Write your answer here…*

**Exercise 3.3** Recommendation Algorithm: Implement the `recommend` function. Identify a suitable, simple recommendation algorithm that will recommend 5 relevant posts on the "Recommended" tab based on the posts the user reacted to positively and the users they followed. (5 points)

*Write your answer here…*

## Task 4 (due 27.10.2025 23:59) 20 points

**Exercise 4.1** Topics: Identify the 10 most popular topics discussed on our platform. Use Latent Dirichlet Allocation (LDA) with the `gensim` library. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. 5 points)

*Write your answer here…*

**Exercise 4.2** Sentiment: Perform sentiment analysis on posts and comments. What is the overall tone of the platform? How does sentiment vary across user posts discussing different topics identified in Exercise 3? Please use VADER (`nltk.sentiment`) for this analysis. Answer and explain your queries/calculations below. You may use SQL and/or Python to perform this task. (5 points)

*Write your answer here…*

**Exercise 4.3** Learning from others' mistakes: Find two social platforms similar to Mini Social that have been under fire for an engineering, design or operation error that severely affected a large group of users. Describe how we can learn from their mistakes and draft up a plan about how Mini Social can be improved learning from their mistakes. You do not need to write code in this exercise unless your plan includes a specific change to an algorithm or function. (5 points)

*Write your answer here…*

**Exercise 4.4** Design and implement a new social feature in Mini Social. For example, a user reputation scoring system, a reporting system, a feature to find related content to a post, new post modalities such as polls or reposts. Your change must include a UI improvement or addition. Do not implement non-social, technical features, such as resource optimization, security improvements or style changes. Document the design and implementation process of your addition here. You must also demonstrate a fully functional feature in a maximum 2-minute video recording uploaded to Moodle. (5 points)

*Write your answer here…*