WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Assignment Cover Sheet

Student Number:	563111, 5662419, 5665831, 5667011, 5667395, 5673899
Module Code:	IB9HP0
Module Title:	Data Management
Submission Deadline:	20 March 2025 12:00:00 (UK time)
Date Submitted:	19 March 2025
Word Count:	1999
Number of Pages:	91
Question Attempted: *(question number/title, or description of assignment)*	Simulate the process of creating a data product for a firm, providing business insights from the database created. Summarize the project in a structured report. Include data visualizations and key insights.
Have you used Artificial Intelligence (AI) in any part of this assignment?	Yes. AI was used in the preparation of this work. It was used in the development of the code: It was used to provide example uses or approaches to elements of the challenges which were then interpreted and modified by the group to be applicable to this data/report. AI was also used for image generation of the hypothetical firm in the presentation.

Academic Integrity Declaration

We're part of an academic community at Warwick. Whether studying, teaching, or researching, we're all taking part in an expert conversation which must meet standards of academic integrity. When we all meet these standards, we can take pride in our own academic achievements, as individuals and as an academic community.

Academic integrity means committing to honesty in academic work, giving credit where we've used others' ideas and being proud of our own achievements.

In submitting my work, I confirm that:

- I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.
- I declare that this work is being submitted on behalf of my group and is all our own, , except where I have stated otherwise.
- No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.
- Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.
- I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.
- Where a proof-reader, paid or unpaid was used, I confirm that the proof-reader was made aware of and has complied with the University's proofreading policy.

Upon electronic submission of your assessment you will be required to agree to the statements above

Table of Contents

1 Introduction

The hypothetical magazine subscription company BABA was created to simulate the development of a data product that provides business insights. This report outlines the data product creation process – establishing the business context, designing and implementing an appropriate database schema, generating synthetic data for the database, and creating business insights from the database.

2 Business Context

BABA is a UK subscription magazine company, offering monthly plans at various price points to customers. Key business functions include revenue management, subscriptions management, and content management. Revenue management includes subscription payments and revenue forecasting, while subscriptions management requires subscription plan oversight and user behavior analysis. Content management considers content performance. The database mini world, a simplified representation of core data related to the business, should focus on the relationships between components related to these key business functions. Thus, elements such as users, subscriptions, subscription plans, payments, and content should be included.

The purpose of the data product generated from the mini world is to provide BABA with actionable business insights into user behaviour, subscription trends, and content consumption. These insights could in turn help the firm make data driven decisions to optimize revenue, customer retention, and content delivery. Expected insights from the database include reports on subscription plan performance, user engagement, customer behaviour by demographic, and payment method preferences.

3 Database Design and Implementation

3.1 Database Design

Based on the business context identified, an Entity-Relationship (ER) Diagram was created to represent the data model as seen in Figure 1.

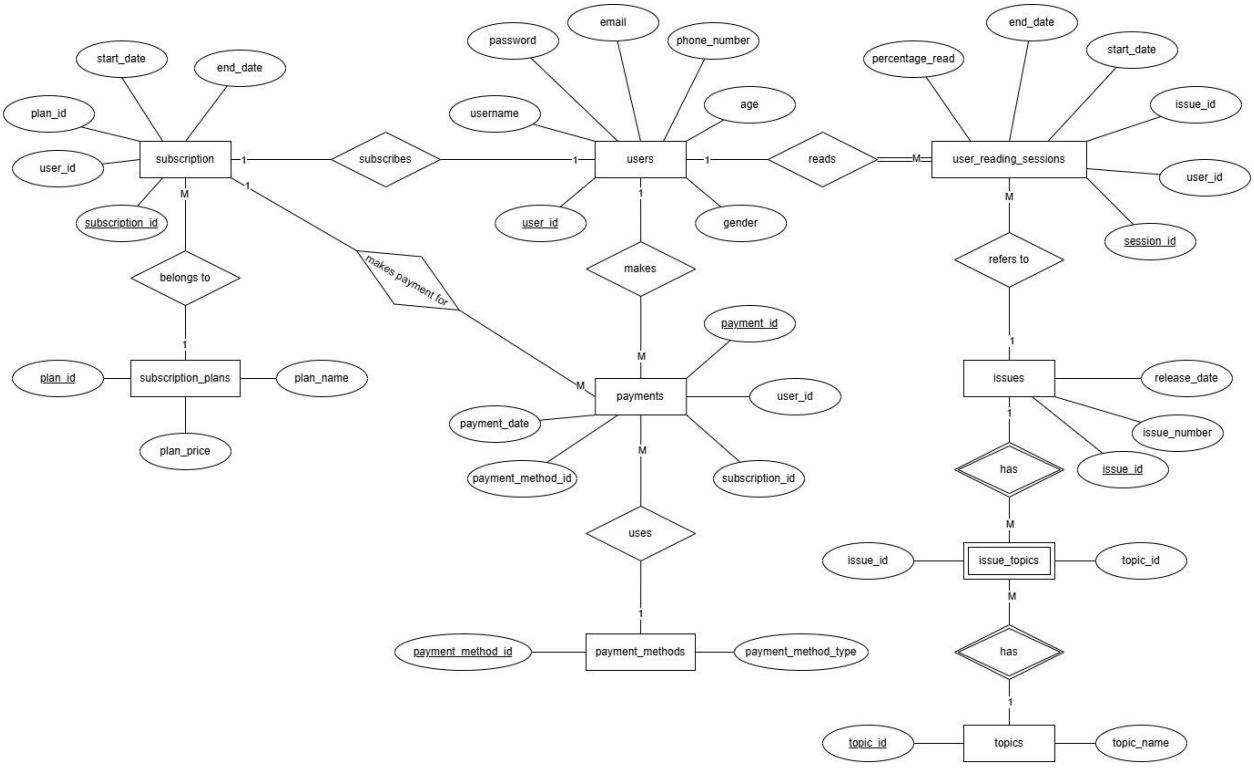

Figure 1. CHEN's Notation: Entity-Relationship (ER) Diagram

The database has nine entities – subscription_plans, subscriptions, users, payment_methods, payments, issues, topics, issue_topics, and user_reading_sessions. subscription_plans has attributes plan_id, plan_name, and plan_price. plan_id identifies each unique subscription plan BABA offers and is the entity's primary key (PK), used to identify a particular entity and inherently unique and not null. Attributes plan_name and plan_price provide additional information on each plan and are unique and not null.

The subscriptions entity has subscription_id, user_id, plan_id, start_date, and end_date. subscription_id is the PK, documenting each unique subscription. start_date and end_date give the beginning and end date of subscriptions. start_date is not null, while end_date is null for active subscriptions. user_id is a foreign key (FK) and is unique, enforcing a 1:1 relationship where each user has one auto-renewing subscription, preventing plan changes or resubscriptions. plan_id is also a FK but is not unique, allowing multiple subscriptions under the same plan, forming an M:1 relationship with subscription_plans.

The users entity has attributes user_id, username, email, phone_number, age, and gender. user_id is a PK, identifying each unique customer. Other entity attributes provide additional

information on each user and cannot be null, with each username, email, and phone_number also being unique.

The payment_methods entity contains attributes payment_method_id and payment_method_type. payment_method_id is the PK, showing each unique payment method BABA accepts. payment_method_type is not null and unique, defining payment method types.

Attributes payment_id, user_id, subscription_id, payment_method_id, and payment_date belong to payments. payment_id is the PK and documents each payment made to BABA. payment_date is not null and represents when the payment was made. user_id, subscription_id, and payment_method_id are FKs which cannot be null but are also not unique. For user_id and subscription_id, this is because the users and subscriptions entities both have 1:M relationships with payments, where one user with one subscription can make multiple payments. payment_method_id comes from the payment_methods entity, which has M:1 relationship with payments. This means many payments can have the same payment type.

The entity issues have the attributes issue_id, issue_number, and release_date. issue_id is the PK and documents each BABA magazine issue. issue_number and release_date are both unique and not null, providing additional information on issues. The entity topics contains attributes topic_id and topic_name, where topic_id is the PK, representing each unique magazine topic. topic_name defines each topic and is unique and not null. The entities issues and topics have a M:N relationship, where a magazine issue can have multiple topics, and a topic can be tagged for multiple issues. To maintain data integrity, the weak entity issue_topics, which cannot be uniquely identified by its attributes alone, was created. The attributes of the entity are issue_id and topic_id, FKs that combine to create a composite PK. These values are not null but not unique to fulfil the M:N relationship condition. However, an issue_id cannot have multiple entries with the same topic_id, as this would create redundancy.

The user_reading_sessions stores reading sessions for BABA magazine, with session_id as the PK. Attributes include non-unique FKs user_id and issue_id, start_date, end_date, and percentage_read. start_date, not null, marks when a user begins reading, while end_date is recorded upon completion or remains null if unfinished. percentage_read is not null and tracks reader progress. Users have a 1:M (0..M) relationship with sessions, meaning a user may have multiple or no sessions, but each session must have a user. Issues have a 1:M relationship with sessions, allowing multiple readings per issue. However, a user cannot log multiple sessions for the same issue_id, as end_date tracks completion, preventing rereads.

3.2 SQL Schema Implementation

Upon completing database design, the schema was implemented in SQLite with all constraints, keys, and relationships identified from the ER diagram. The implementation required the definition of data types for attributes.

SQLite has four datatypes: TEXT, INTEGER, REAL, BLOB. TEXT stores character strings, INTEGER stores integer values, REAL stores floating-point numbers, and BLOB stores unstructured data. For attributes that were strings, TEXT was assigned. This includes primary keys like user_id, descriptive attributes like plan_name, and also dates as SQLite lacks a date datatype. REAL was used for attributes with decimals like percentage_read and plan_price. INTEGER was assigned to numeric whole values like phone_number, age, and plan_id. BLOB was not used as there were no attributes with unstructured data. For attributes with a set of named values like gender and plan_name, a CHECK constraint was also added to ensure data integrity. Refer to Appendix A for the data dictionary and Appendix B for the database creation code.

4 Data Generation and Importation

4.1 Synthetic Data Generation

Data was generated synthetically to populate the database following schema implementation using Python's Faker library. The aim was to produce data that reflected realistic business operations. Data assumptions and constraints were made to provide actionable business are discussed below.

Regarding constraints, data was generated within the parameters of the database design. This ensured entity relationship conditions such as each user having one subscription and one user having from zero to multiple reading sessions were met. Logical constraints were also implemented for dates and topics. The first payment a user makes coincides with the day their subscription starts, and a payment is made every subsequent month until the subscription ends. Reading sessions must be within the subscription period, while the issue being read must be released before the reading session begins. A magazine issue can be assigned to at most three topics.

Various assumptions were also made about the data to create realistic conditions. The overall distributions of three key components – gender, age, and subscription plan – were all adjusted. The male and female categories were made to be much larger than other gender categories. Age was normally distributed and certain subscription plans were more prevalent than others.

Subscription plan preferences were also adjusted to differ by gender and age, while average subscription duration changed based on all key elements. Seasonality was introduced to subscriptions, making cancellations more common in certain months. Topic popularity varies depending on age and gender, and reading sessions differed in all key components. The distribution of payment methods used was manipulated to make some payment methods more ubiquitous generally and also changed so that payment method preference changed depending on age. Refer to Appendix C for the data generation code.

4.2 Database Population

The data was loaded into the relational database after generation, revealing some issues with the database created and generated data. The absence of certain unique constraints in data generation and inconsistencies in applying unique constraints from the ER diagram across tables in database creation caused several issues during the data import process. Some data could not be imported as they violated database constraints, while other data with issues were still imported because of incorrect database creation code. This led to potential duplicate entries during the import process. To resolve this, unique constraints were applied to user_id, subscription_id, payment_id, and issue_id in the relevant tables, and data generation code was revised to meet database constraints. Example rows from the tables after successful data import are illustrated in Figure 2–10. Refer to Appendix B for the data population code.

	plan_id	plan_name	plan_price
1	1	Basic	£5.99
2	2	Standard	£12.99
3	3	Premium	£18.99

Figure 2. Subscription Plans Table

	subscription_id	user_id	plan_id	start_date	end_date
1	JBQ6H	66jM2	1	2022-12-23	2023-12-23
2	5X5X4	iEGtN	1	2020-10-22	2025-02-22
3	9ETPQ	HKUw8	1	2024-12-26	2025-02-26
4	81VQK	qGpSe	2	2023-05-20	None
5	7FPCL	ms6WG	1	2018-03-28	None
6	W08F6	UwaUU	2	2021-01-22	None
7	MADD9	ZiZqp	2	2018-10-26	None
8	CPDE3	oQdVl	3	2020-09-14	None
9	P9LGK	tuLar	3	2021-03-27	2023-04-27
10	2WHY5	ThxbH	3	2018-03-05	2025-02-05

Figure 3. Subscriptions Table

	user_id	username	password	email	phone_number	age	gender
1	66jM2	johnsonjoshua	cs&2#4Xd$a	johnsonjoshua@hotmail.com	449932406875	33	male
2	iEGtN	lisa02	+6C&Koil+R	lisa02@hotmail.com	444936280471	44	female
3	HKUw8	barbara10	y3YGLn0E_j	barbara10@hotmail.com	443162783940	23	female
4	qGpSe	frankgray	Bi_8lHHvTf	frankgray@yahoo.com	444572801346	30	male
5	ms6WG	johnhoffman	H%zP&$N9ks	johnhoffman@hotmail.com	446947632501	26	male
6	UwaUU	tasha01	i89+Yffx_i	tasha01@gmail.com	443497126385	25	female
7	ZiZqp	nathanielmartin	O0Tyl55k(g	nathanielmartin@hotmail.com	444158067293	20	male
8	oQdVl	josephbrennan	y2BRaKQ(&3	josephbrennan@gmail.com	444796452830	36	female
9	tuLar	perezrebecca	^NRfr1QhI&	perezrebecca@hotmail.com	443371905468	18	male
10	ThxbH	brownjessica	3g1Gq^lp@a	brownjessica@gmail.com	445341978065	25	female

Figure 4. Users Table

	payment_method_id	payment_method_type
1	1	credit_card
2	2	debit_card
3	3	gift_card
4	4	paypal
5	5	apple_pay
6	6	google_pay
7	7	bank_transfer

Figure 5. Payment Methods Table

	payment_id	user_id	subscription_id	payment_method_id	payment_date
1	fqdl370	66jM2	JBQ6H	1	2022-12-23
2	357lv65	66jM2	JBQ6H	5	2023-01-23
3	17ldulj	66jM2	JBQ6H	5	2023-02-23
4	ykv4hdp	66jM2	JBQ6H	3	2023-03-23
5	yh7aziw	66jM2	JBQ6H	1	2023-04-23
6	ld13m78	66jM2	JBQ6H	6	2023-05-23
7	b2l6web	66jM2	JBQ6H	1	2023-06-23
8	6h8gvlb	66jM2	JBQ6H	7	2023-07-23
9	w51w6mg	66jM2	JBQ6H	6	2023-08-23
10	8k2wwsl	66jM2	JBQ6H	2	2023-09-23

Figure 6. Payments Table

	issue_id	issue_number	release_date
1	EZ2D	1	2017-01-03
2	SZ4O	2	2017-01-10
3	ULA3	3	2017-01-21
4	5ZCT	4	2017-01-24
5	7GLM	5	2017-02-06
6	3P8A	6	2017-02-07
7	1T5D	7	2017-02-12
8	V7A2	8	2017-02-22
9	T7BJ	9	2017-02-23
10	S0XJ	10	2017-02-25

Figure 7. Issues Table

	topic_id	topic_name
1	1	Business & Finance
2	2	Technology & Innovation
3	3	Books & Literature
4	4	Health & Wellness
5	5	DIY & Crafting
6	6	Relationships & Dating
7	7	Fashion & Beauty
8	8	Travel & Adventure
9	9	Food & Cooking
10	10	Parenting & Family

Figure 8. Topics Table

	issue_id	topic_id
1	EZ2D	2
2	EZ2D	11
3	EZ2D	14
4	SZ4O	4
5	SZ4O	6
6	ULA3	1
7	ULA3	2
8	ULA3	3
9	5ZCT	9
10	5ZCT	12

Figure 9. Issues Topic Table

	session_id	user_id	issue_id	start_date	end_date	percentage_read
1	0znunx	66jM2	4LCJ	2023-10-16	2023-12-23	32.99
2	x7szku	iEGtN	83CI	2023-04-30	2025-02-22	71.94
3	3r4cgl	iEGtN	NC2V	2024-10-06	2025-02-22	100.00
4	3u4t14	HKUw8	5HNO	2025-02-24	2025-02-26	100.00
5	lh432a	HKUw8	2UM0	2025-02-18	2025-02-26	100.00
6	f75f7c	qGpSe	NM2A	2025-02-18		25.54
7	5b83wu	qGpSe	1HAD	2025-02-17		48.30
8	9ryeg5	qGpSe	4WCV	2024-11-20		40.88
9	c8t9ps	ms6WG	70DK	2022-12-15		5.09
10	xlpm4n	UwaUU	NM2A	2024-10-18		100.00

Figure 10. User Reading Sessions Table

5 Business Insights

After populating the database, SQLite queries were used to produce business insights. Refer to Appendix D for the business insights code.

5.1 User Distribution

The user distribution analysis shown in Figure 11–13 highlight significant demographic trends. The 25-34 group is the largest, followed by 18-24, with engagement dropping after 35. Female users lead, followed by males and non-binary users. Standard and Premium plans have similar users, while Basic is the lowest. Highlighting a younger audience favoring higher-tier plans, requiring tailored retention strategies.

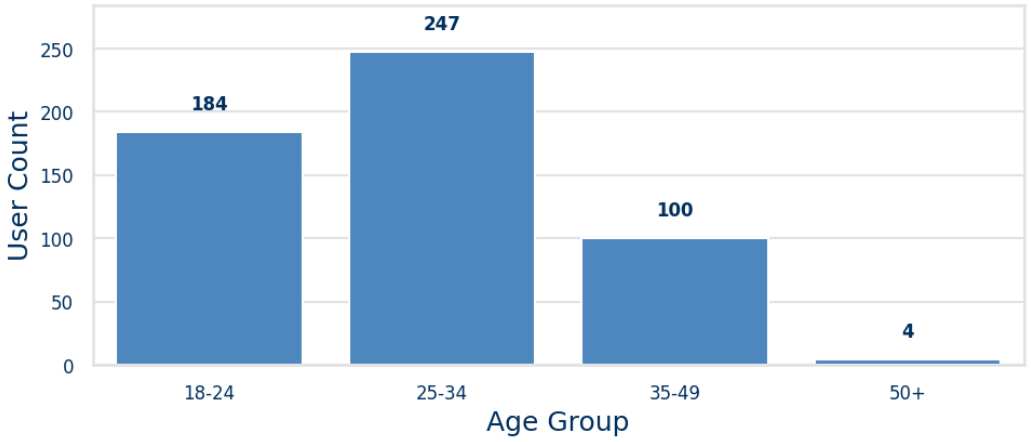

Figure 11. Number of Users by Age Group

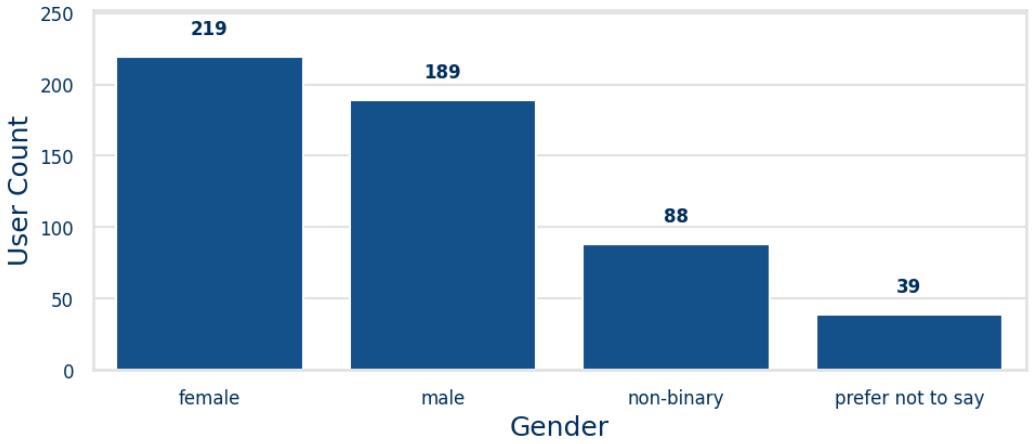

Figure 12. Number of Users by Gender

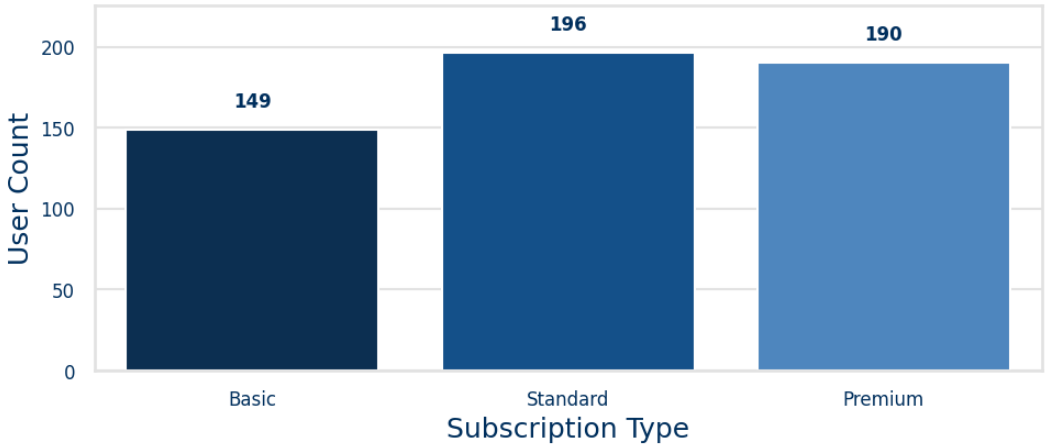

Figure 13. Number of Users by Subscription Types

5.2 Payment Methods

Apple Pay leads in usage, followed by Google Pay and credit cards, while debit cards, PayPal, and bank transfers are less common. Younger users (18-34) prefer mobile payments, whereas older demographics favor credit cards and PayPal. Optimizing mobile payment options and enhancing digital wallet compatibility can improve transaction efficiency and customer satisfaction.

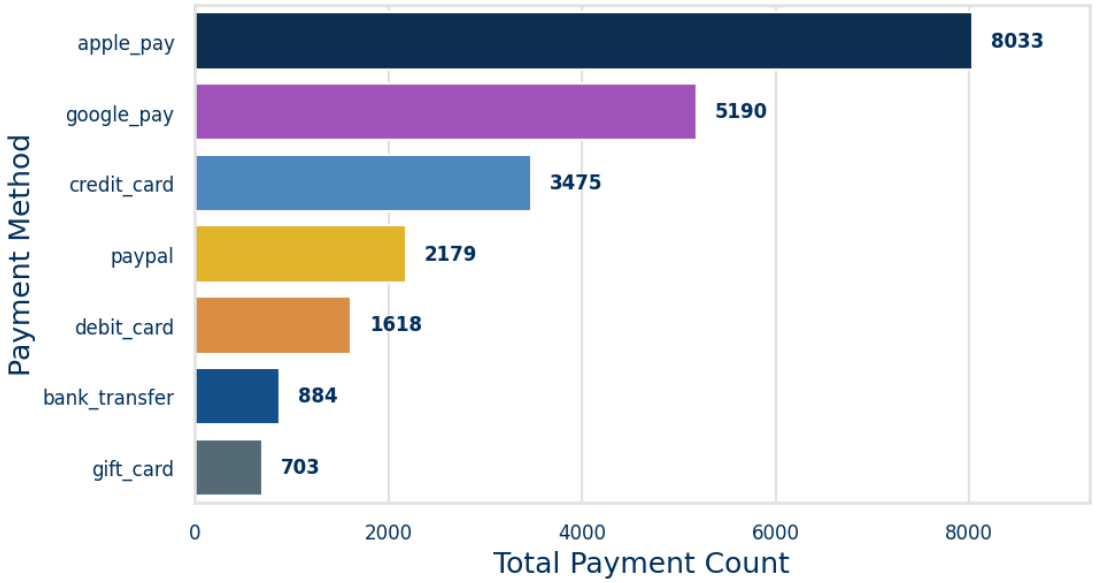

Figure 14. Most Frequently Used Payment Methods

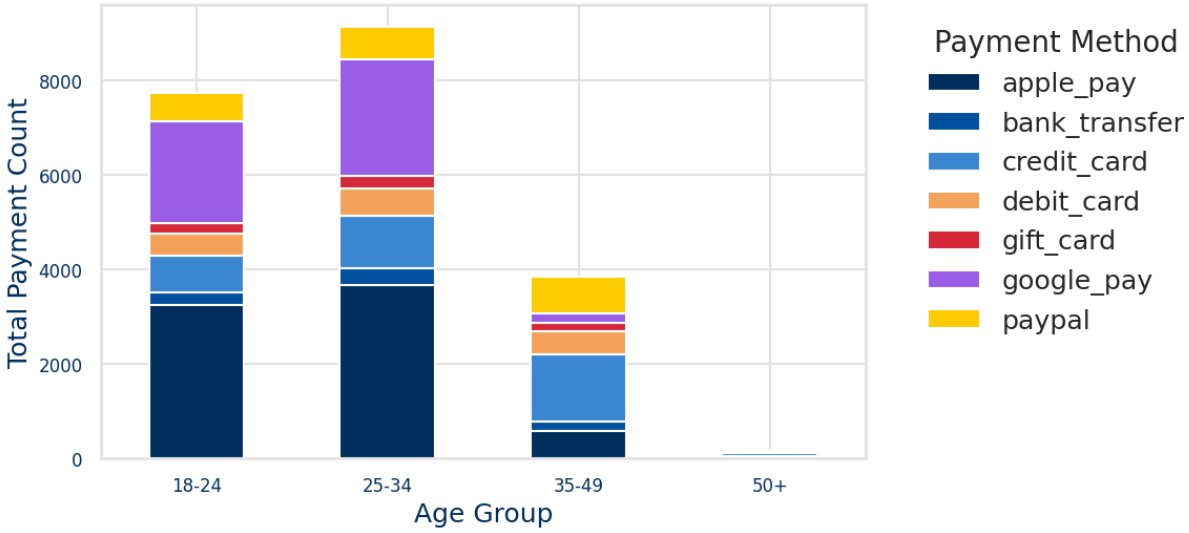

Figure 15. Payment Method Usage by Age Group

5.3 Revenue and Churn Rates

Premium plans generate the highest revenue, followed by Standard and Basic. Female users contribute the most revenue. Churn analysis shows Premium has the lowest churn, while Basic has the highest. Higher-tier plans boost revenue and retention, highlighting the need to promote Premium subscriptions for sustained growth and long-term customer engagement.

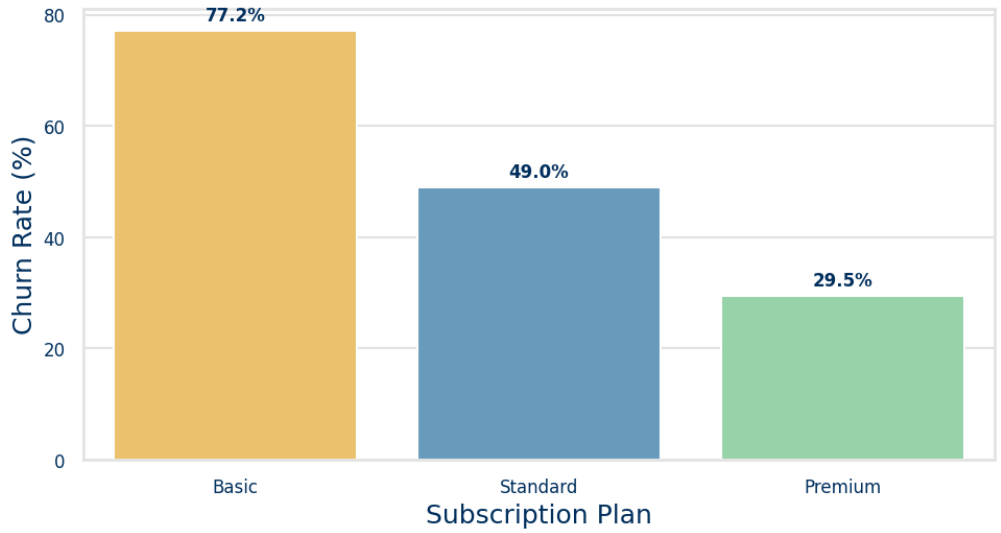

Figure 16. Churn Rate by Subscription Plan

Figure 17. Total Revenue by Subscription Plan

5.4 Content

Business & Finance is the most popular topic, reflecting strong interest in financial literacy. Fashion & Beauty, Health & Wellness, and Movies & TV Shows also attract high engagement, while Music & Performing Arts ranks lower. Aligning content strategies with these insights can sustain demand for top topics while boosting engagement in less popular categories.

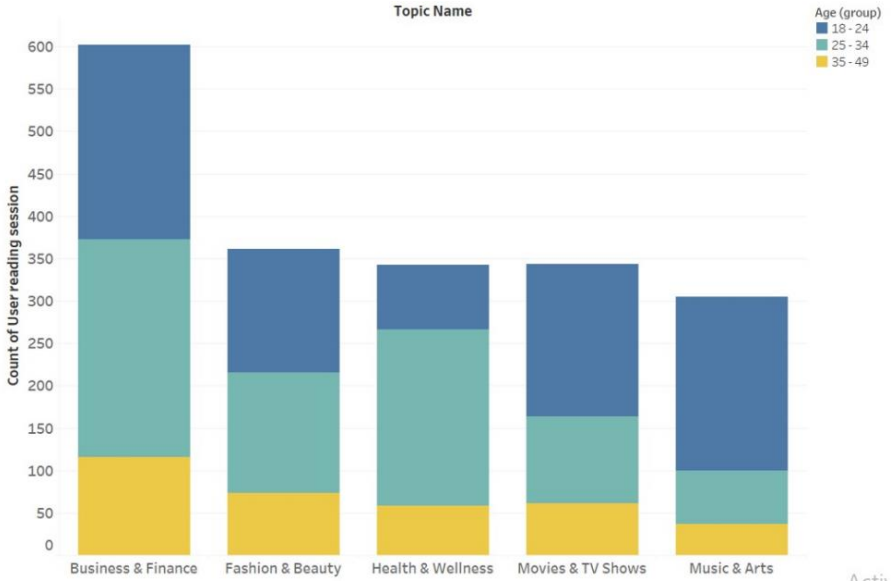

Figure 18. Topic Popularity by Age

5.5 User Engagement

Reading engagement increases with subscription level, with Premium users showing the highest completion rates. More Premium users read over 85% of content compared to Standard and Basic subscribers. These insights highlight the strong value perception among Premium users, emphasizing the need for tailored content strategies to boost engagement among lower-tier subscribers.

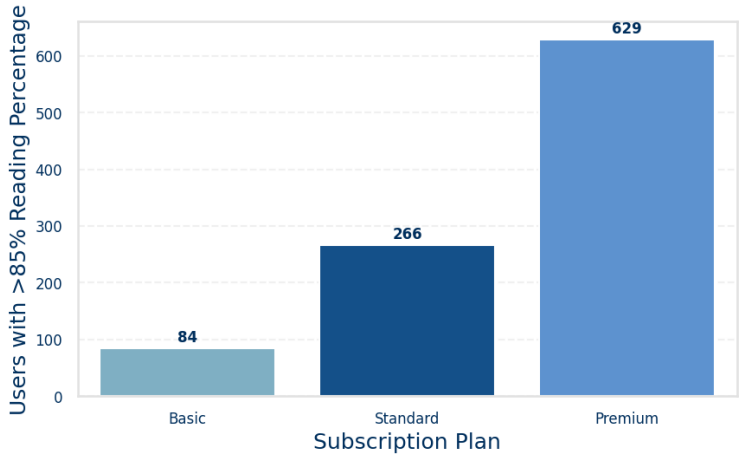

Figure 19. Users Reading More Than 85 Percent of Issues by Subscription Plan

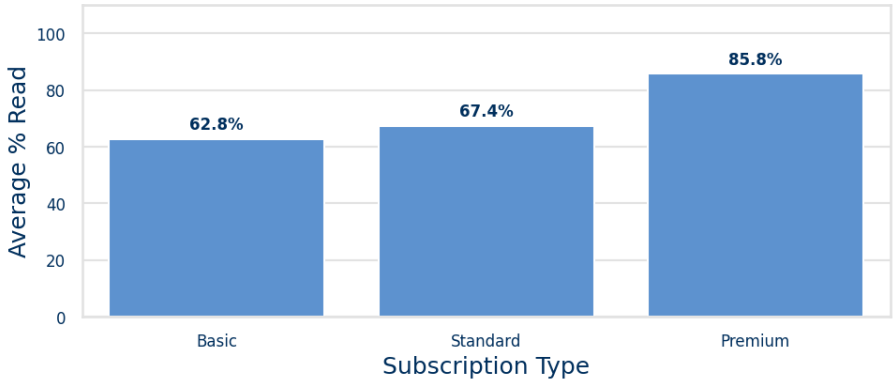

Figure 20. Average Percentage Read by Subscription Types

6 Conclusion

This project simulated the creation of a data product for the magazine company BABA, from initial exploration of the business context to business insights extraction. While this process gave BABA valuable information for strategic decision making, there are some limitations to the database created. As data needed to be synthetically generated for the database, the team opted for a simpler schema design to avoid complex relationships between entities. While this produced a high-quality dataset for analysis, several factors were excluded from business insights. This includes devices which could improve understanding of user behaviors, advertisements for more accurate revenue calculations, and user location to improve customer segmentation. Despite this, the data product still provides critical insights for BABA to optimize its user engagement, maximize revenue, and tailor content delivery. By leveraging the product, BABA can make data-informed business decisions, giving them a strong strategic advantage.

7 Appendices

7.1 Appendix A: Data Dictionary

Appendix A relates to section 3.2 of the report and provides additional information on attributes within the database. The attribute name, SQLite data type, SQLite constraints, and attribute description are given for each entity created in the database.

Column Name	Data Type	Constraints	Description
user_id	TEXT	PRIMARY KEY	Unique identifier for each user
username	TEXT	NOT NULL, UNIQUE	Unique username for the user
password	TEXT	NOT NULL	Encrypted user password
email	TEXT	NOT NULL, UNIQUE	User email address
phone_number	INTEGER	NOT NULL	User phone number
age	INTEGER	NOT NULL, CHECK (age >= 0 AND age <= 100)	Age of the user
gender	TEXT	NOT NULL, CHECK (gender IN ('male', 'female', 'non-binary', 'prefer not to say'))	Gender of the user

Table B1. User Table (users)

Column Name	Data Type	Constraints	Description
plan_id	INTEGER	PRIMARY KEY	Unique identifier for each subscription plan
plan_name	TEXT	NOT NULL, UNIQUE, CHECK (plan_name IN ('Basic', 'Standard', 'Premium'))	Subscription plan type
plan_price	REAL	NOT NULL, UNIQUE, CHECK (plan_price IN ('£5.99', '£12.99', '£18.99'))	Subscription plan price

Table B2. Subscription Plans Table (subscription_plans)

Column Name	Data Type	Constraints	Description
subscription_id	TEXT	PRIMARY KEY	Unique identifier for each subscription
user_id	TEXT	NOT NULL, UNIQUE, FOREIGN KEY	Reference to users.user_id (Subscribed user)
plan_id	INTEGER	NOT NULL, FOREIGN KEY	Reference to subscription_plans.plan_id (Selected plan)
start_date	TEXT	NOT NULL	Subscription start date
end_date	TEXT		Subscription end date (NULL if active).

Table B3. User Subscription Table (subscriptions)

Column Name	Data Type	Constraints	Description
payment_method_id	INTEGER	PRIMARY KEY	Unique identifier for each payment method
payment_method_type	TEXT	NOT NULL, UNIQUE, CHECK (payment_method_type IN ('credit_card', 'gift_card', 'paypal', 'apple_pay', 'google_pay', 'debit_card', 'bank_transfer'))	Payment method type

Table B4. Payment Methods Table (payment_methods)

Column Name	Data Type	Constraints	Description
payment_id	TEXT	PRIMARY KEY	Unique identifier for each payment transaction
user_id	TEXT	NOT NULL, FOREIGN KEY	Reference to users.user_id (Paying user)
subscription_id	TEXT	NOT NULL, FOREIGN KEY	Reference to subscriptions.subscription_id (Associated subscription)
payment_method_id	INTEGER	NOT NULL, FOREIGN KEY	Reference to payment_methods.payment_method_id (Payment method used)
payment_date	TEXT	NOT NULL	The date when the payment was made

Table B5. User Payments Table (payments)

Column Name	Data Type	Constraints	Description
issue_id	TEXT	PRIMARY KEY	Unique identifier for each magazine issue
issue_number	INTEGER	NOT NULL, UNIQUE	The issue number
release_date	TEXT	NOT NULL, UNIQUE	The publication date of the issue.

Table B6. Magazine Issues Table (issues)

Column Name	Data Type	Constraints	Description
topic_id	INTEGER	PRIMARY KEY	Unique identifier for each topic
topic_name	TEXT	NOT NULL, UNIQUE	Name of the topic

Table B7. Topics Table (topics)

Column Name	Data Type	Constraints	Description
issue_id	TEXT	NOT NULL, FOREIGN KEY, PRIMARY KEY	Reference to issues.issue_id (Magazine issue)
topic_id	INTEGER	NOT NULL, FOREIGN KEY, PRIMARY KEY	Reference to topics.topic_id (Topic category)

Table B8. Issue–Topics Mapping Table (issue_topics)

Column Name	Data Type	Constraints	Description
sesson_id	TEXT	PRIMARY KEY	Unique identifier for each reading session
user_id	TEXT	NOT NULL, FOREIGN KEY	Reference to users.user_id (User reading the issue)
issue_id	TEXT	NOT NULL, FOREIGN KEY	Reference to issues.issue_id (Magazine issue being read)
start_date	TEXT	NOT NULL	When the user started reading this issue
end_date	TEXT		When the user finished reading the issue (NULL if issue is not completed)
percentage_read	REAL	NOT NULL, DEFAULT 0.00, CHECK (percentage_read >= 0 AND percentage_read <= 100)	The percentage of the issue read (0.00 - 100.00)

Table B9. User Reading Progress Table (user_reading_sessions)

7.2 Appendix B: Database Generation Code

Appendix B relates to section 3.2 and 4.2 of the report, providing the code used to generate and populate the relational database in SQLite.

Step 1. Create the SQLite Database

```
# Load libraries
import sqlite3

def create_database():
    # Connect to SQLite database
    conn = sqlite3.connect('magazine.db')
    cursor = conn.cursor()

    # Enable foreign keys support in SQLite
    cursor.execute("PRAGMA foreign_keys = ON;")


    # Table 1. Create users
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        user_id TEXT PRIMARY KEY,
        username TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE,
        phone_number INTEGER NOT NULL UNIQUE,
        age INTEGER NOT NULL CHECK (age >= 0 AND age <= 100),
         gender TEXT NOT NULL CHECK (gender IN ('male', 'female', 'non-binary',
'prefer not to say'))
    )
    ''')


    # Table 2. Create subscription_plans
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS subscription_plans (
        plan_id INTEGER PRIMARY KEY,
        plan_name TEXT NOT NULL UNIQUE CHECK (plan_name IN ('Basic', 'Standard',
'Premium')),
        plan_price REAL NOT NULL UNIQUE CHECK (plan_price IN ('£5.99', '£12.99',
'£18.99'))
    )
    ''')


    # Table 3. Create subscriptions
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS subscriptions (
        subscription_id TEXT PRIMARY KEY,
        user_id TEXT UNIQUE,
        plan_id INTEGER NOT NULL,
```

```
        start_date TEXT NOT NULL,
        end_date TEXT,
        FOREIGN KEY (user_id) REFERENCES users(user_id),
        FOREIGN KEY (plan_id) REFERENCES subscription_plans(plan_id)
    )
    ''')


    # Table 4. Create payment_methods
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS payment_methods (
        payment_method_id INTEGER PRIMARY KEY,
         payment_method_type TEXT NOT NULL UNIQUE CHECK (payment_method_type IN
('credit_card', 'gift_card', 'paypal', 'apple_pay', 'google_pay', 'debit_card',
'bank_transfer'))
    )
    ''')


    # Table 5. Create payments
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS payments (
        payment_id TEXT PRIMARY KEY,
        user_id TEXT NOT NULL,
        subscription_id TEXT NOT NULL,
        payment_method_id INTEGER NOT NULL,
        payment_date TEXT NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users(user_id),
       FOREIGN KEY (subscription_id) REFERENCES subscriptions(subscription_id),
                          FOREIGN     KEY     (payment_method_id)     REFERENCES
payment_methods(payment_method_id)
    )
    ''')


    # Table 6. Create issues
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS issues (
        issue_id TEXT PRIMARY KEY,
        issue_number INTEGER NOT NULL UNIQUE,
        release_date TEXT NOT NULL UNIQUE
    )
    ''')


    # Table 7. Create topics
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS topics (
        topic_id INTEGER PRIMARY KEY,
        topic_name TEXT NOT NULL UNIQUE
    )
    ''')
```

```
    # Table 8. Create issue_topics
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS issue_topics (
        issue_id TEXT NOT NULL,
        topic_id INTEGER NOT NULL,
        PRIMARY KEY (issue_id, topic_id),
        FOREIGN KEY (issue_id) REFERENCES issues(issue_id),
        FOREIGN KEY (topic_id) REFERENCES topics(topic_id)
    )
    ''')


    # Table 9. Create user_reading_sessions
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS user_reading_sessions (
        session_id TEXT PRIMARY KEY,
        user_id TEXT NOT NULL,
        issue_id TEXT NOT NULL,
        start_date TEXT NOT NULL,
        end_date TEXT,
       percentage_read REAL NOT NULL DEFAULT 0.00 CHECK (percentage_read >= 0 AND
percentage_read <= 100),
        FOREIGN KEY (user_id) REFERENCES users(user_id),
        FOREIGN KEY (issue_id) REFERENCES issues(issue_id)
    )
    ''')



    # Commit and close
    conn.commit()
    conn.close()
    print("Database and tables created successfully.")

# Run the function to create the database
create_database()
```

Step 2. Check Tables Created

```
# Load libraries
import sqlite3

# Reconnect to the database
conn = sqlite3.connect('magazine.db')
cursor = conn.cursor()

# Get all table names
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

# Traverse the table and get column information
for table_name in tables:
    print(f"Table: {table_name[0]}")
    cursor.execute(f"PRAGMA table_info({table_name[0]});")
    columns = cursor.fetchall()
    for col in columns:
        print(f"  Column: {col[1]}, Type: {col[2]}, NotNull: {col[3]},
DefaultVal: {col[4]}, PrimaryKey: {col[5]}")
    print("-" * 20)

# Close connection
conn.close()
```

Step 3. Upload Files

```
# Upload csv files
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

Step 4. Load Files into Database Tables

```
import sqlite3
import csv

# connect database
conn = sqlite3.connect('magazine.db')
cursor = conn.cursor()

def import_csv_to_table(csv_file, table_name):
    try:
        with open(csv_file, 'r', encoding='utf-8') as file:
            csv_reader = csv.reader(file)
            next(csv_reader)
            for row in csv_reader:
                placeholders = ', '.join(['?' for _ in row])
                sql = f"INSERT INTO {table_name} VALUES ({placeholders})"
                cursor.execute(sql, row)
    except Exception as e:
        print(f"Error importing {csv_file} into {table_name}: {e}")
        conn.rollback()

# Make sure the database connection is open
try:
    import_csv_to_table('users.csv', 'users')
    import_csv_to_table('subscription_plans.csv', 'subscription_plans')
    import_csv_to_table('subscriptions.csv', 'subscriptions')
    import_csv_to_table('payment_methods.csv', 'payment_methods')
    import_csv_to_table('payments.csv', 'payments')
    import_csv_to_table('issues.csv', 'issues')
    import_csv_to_table('topics.csv', 'topics')
    import_csv_to_table('issue_topics.csv', 'issue_topics')
    import_csv_to_table('user_reading_sessions.csv', 'user_reading_sessions')

    conn.commit()

    print("Data imported successfully!")
except Exception as e:
    print(f"An error occurred: {e}")
    conn.rollback()
finally:
    cursor.close()
    conn.close()
```

Step 5. Check Database Data

```
# Load libraries
import pandas as pd
import sqlite3

# Connect database
conn = sqlite3.connect('magazine.db')
cursor = conn.cursor()

# Query all tables and load into pandas DataFrames
users_df = pd.read_sql_query("SELECT * FROM users", conn)
subscription_plans_df = pd.read_sql_query("SELECT * FROM subscription_plans",
conn)
subscriptions_df = pd.read_sql_query("SELECT * FROM subscriptions", conn)
payment_methods_df = pd.read_sql_query("SELECT * FROM payment_methods", conn)
payments_df = pd.read_sql_query("SELECT * FROM payments", conn)
issues_df = pd.read_sql_query("SELECT * FROM issues", conn)
topics_df = pd.read_sql_query("SELECT * FROM topics", conn)
issue_topics_df = pd.read_sql_query("SELECT * FROM issue_topics", conn)
user_reading_sessions_df      =      pd.read_sql_query("SELECT      *      FROM
user_reading_sessions", conn)

# Create a new index starting from 1
user_new_index = pd.RangeIndex(start=1, stop=len(users_df) + 1)
subscription_plans_new_index              =              pd.RangeIndex(start=1,
stop=len(subscription_plans_df) + 1)
subscriptions_new_index = pd.RangeIndex(start=1, stop=len(subscriptions_df) +
1)
payment_methods_new_index = pd.RangeIndex(start=1, stop=len(payment_methods_df)
+ 1)
payments_new_index = pd.RangeIndex(start=1, stop=len(payments_df) + 1)
issues_new_index = pd.RangeIndex(start=1, stop=len(issues_df) + 1)
topics_new_index = pd.RangeIndex(start=1, stop=len(topics_df) + 1)
issue_topics_new_index  =  pd.RangeIndex(start=1,  stop=len(issue_topics_df)  +
1)
user_reading_sessions_new_index            =            pd.RangeIndex(start=1,
stop=len(user_reading_sessions_df) + 1)

# Assign the new index to each DataFrame
users_df.index = user_new_index
subscription_plans_df.index = subscription_plans_new_index
subscriptions_df.index = subscriptions_new_index
payment_methods_df.index = payment_methods_new_index
payments_df.index = payments_new_index
issues_df.index = issues_new_index
topics_df.index = topics_new_index
issue_topics_df.index = issue_topics_new_index
user_reading_sessions_df.index = user_reading_sessions_new_index

# Show the first 10 lines of each DataFrame
```

```
print("\nUsers Table:")
print(users_df.head(10).to_string())

print("\nSubscription plans Table:")
print(subscription_plans_df.head(10).to_string())

print("\nSubscriptions Table:")
print(subscriptions_df.head(10).to_string())

print("\nPayment methods Table:")
print(payment_methods_df.head(10).to_string())

print("\nPayment Table:")
print(payments_df.head(10).to_string())

print("\nIssues Table:")
print(issues_df.head(10).to_string())

print("\nTopics Table:")
print(topics_df.head(10).to_string())

print("\nIssue Topics Table:")
print(issue_topics_df.head(10).to_string())

print("\nUser reading session Table:")
print(user_reading_sessions_df.head(10).to_string())

# Close connection
cursor.close()
conn.close()
```

7.3 Appendix C: Data Generation Code

Appendix C relates to section 4.1 of the report and provides the Python code that was used to generate synthetic data for the relational database.

```
# Load libraries
import pandas as pd
import random
from faker import Faker
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta
import string
import sqlite3

# Create a Faker instance to generate random data
fake = Faker()
Faker.seed(42)

# Create sets to ensure uniqueness
user_ids = set()
usernames = set()
emails = set()
phone_numbers = set()

# Generate a unique user ID
def generate_unique_id():
    while True:
        uid  =  ''.join(random.choices(string.ascii_letters  +  string.digits,
k=5))
        if uid not in user_ids:
            user_ids.add(uid)
            return uid

# Generate a unique username
def generate_unique_username():
    while True:
        uname = fake.user_name()
        if uname not in usernames:
            usernames.add(uname)
            return uname

# Generate a unique email
def generate_unique_email(username):
    while True:
```

```
        # Generate a random email by appending the username with a random
domain
        mail = f"{username}@{fake.free_email_domain()}"
        if mail not in emails:
            emails.add(mail)
            return mail

# Generate a unique phone number
def generate_unique_phone():
    max_attempts = 10000
    attempts = 0

    while attempts < max_attempts:
        second_digit = random.randint(1, 9)  # Randomly select a digit between
1 and 9
        remaining = ''.join(random.sample(string.digits, 9))   # Generate 9
random digits (avoiding duplicates)
        phone_number = f"44{second_digit}{remaining}"   # Format the phone
number with '44' country code

        if phone_number not in phone_numbers:
            phone_numbers.add(phone_number)
            return phone_number

        attempts += 1

    raise RuntimeError("Exceeded phone number attempts. Too many duplicates.")

# Generate an age value using a normal distribution with a mean of 25 and a
standard deviation of 10. Ensures the age falls within 18 to 70.
def generate_age():
    while True:
        age = int(random.gauss(25, 10))
        if 18 <= age <= 70:
            return age

# Select a subscription plan based on gender and age
def select_plan(gender, age):
    if gender == "female":
        plan_weights = [12, 22, 66]
    else:
        plan_weights = [31, 48, 21]

    # Adjust the plan distribution based on age
    if age < 25:
```

```
        plan_weights = [12, 25, 63]
    elif age < 35:
        plan_weights = [37, 38, 25]

    # Randomly select a plan based on the adjusted weights
    return random.choices([1, 2, 3], weights=plan_weights, k=1)[0]

# Function to get the duration of a subscription plan based on gender and
selected plan
def get_subscription_duration(gender, plan_id):
    # Define plan durations and weights for females
    if gender == "female":
        plan_durations = {1: [36, 42, 48, 54], 2: [48, 60, 72], 3: [60, 72,
84]}
        weights = {1: [1,2,4,6], 2: [1,2,7], 3: [1,6,10]}
    # Define plan durations and weights for males
    elif gender == "male":
        plan_durations = {1: [1, 3, 6], 2: [12, 18, 20], 3: [18, 24, 36]}
        weights = {1: [6,3,1], 2: [4,3,2], 3: [5,3,2]}
    # Define plan durations and weights for non-binary and prefer not to say
    else:
        plan_durations = {1: [12, 18, 24], 2: [20,22,24], 3: [36,42,40]}
        weights = {1: [2,3,4], 2: [2,3,5], 3: [1,3,6]}
    return                              random.choices(plan_durations[plan_id],
weights=weights[plan_id])[0]

# Initialise the list to store user information and a list for user IDs
users = []
user_id_list = []

# Generate 535 users
for _ in range(535):
    uid = generate_unique_id()
    user_id_list.append(uid)
    uname = generate_unique_username()
    users.append({
        "user_id": uid,
        "username": uname,
        "password": fake.password(),
        "email": generate_unique_email(uname),
        "phone_number": generate_unique_phone(),
        "age": generate_age(),
        "gender": random.choices(
            ["female","male","non-binary","prefer not to say"],
            weights=[48,31,15,6], k=1
```

```
        )[0]
    })

# Define the number of subscriptions, issues, and sessions
num_subscriptions = 535
num_issues = 520
num_sessions = 1500

# Define the date range for the start and end dates of subscriptions and issues
start_date_range = datetime(2017, 1, 1)
end_date_range = datetime(2025, 2, 28)

# Initialise empty lists to store subscriptions, payments, issues, reading
sessions, and issue-topic mappings
subscriptions = []
payments = []
issues = []
reading_sessions = []
issue_topics = []

# Initialise sets to keep track of unique IDs for subscriptions, payments,
issues, and sessions
subscription_ids = set()
payment_ids = set()
issue_ids = set()
session_ids = set()

# Generate a unique subscription ID
def generate_subscription_id():
    while True:
        sid = ''.join(random.choices(string.ascii_uppercase + string.digits,
k=5))
        if any(c.isalpha() for c in sid) and any(c.isdigit() for c in sid):
            if sid not in subscription_ids:
                subscription_ids.add(sid)
                return sid

# Generate a unique payment ID
def generate_payment_id():
    while True:
        pid = ''.join(random.choices(string.ascii_lowercase + string.digits,
k=7))
        if any(c.isalpha() for c in pid) and any(c.isdigit() for c in pid):
            if pid not in payment_ids:
                payment_ids.add(pid)
```

```
                return pid

# Generate a unique issue ID
def generate_issue_id():
    while True:
        letters = random.choices(string.ascii_uppercase, k=1)
        digits = random.choices(string.digits, k=1)
        remaining = random.choices(string.ascii_uppercase + string.digits, k=2)
        iid = ''.join(random.sample(letters + digits + remaining, 4))
        if iid not in issue_ids:
            issue_ids.add(iid)
            return iid

# Generate a unique session ID
def generate_session_id():
    while True:
        sess_id     =     ''.join(random.choices(string.ascii_lowercase     +
string.digits, k=6))
        if any(c.isalpha() for c in sess_id) and any(c.isdigit() for c in
sess_id):
            if sess_id not in session_ids:
                session_ids.add(sess_id)
                return sess_id

# Define payment methods and their corresponding weights for selection
payment_methods = [1,2,3,4,5,6,7]
payment_weights = [41,8,3,10,30,5,3]

# Select the minimum between num_subscriptions and the total number of users
to ensure we don't exceed the available users
num_subscriptions = min(num_subscriptions, len(user_id_list))
# Randomly sample the user IDs to ensure unique subscriptions
selected_user_ids = random.sample(user_id_list, num_subscriptions)

# Loop through the selected user IDs to generate subscriptions
for uid in user_id_list[:num_subscriptions]:
    subscription_id = generate_subscription_id()
    user_info = next(u for u in users if u["user_id"] == uid)
    gender = user_info["gender"]
    age = user_info["age"]
    plan_id = select_plan(gender, age)

    while True:
        # Randomly generate a start date for the subscription
        ry = random.randint(2017, 2024)
```

```
        rm = random.randint(1, 12)
        rd = random.randint(1, 28)
        start_date = datetime(ry, rm, rd)
        # Get the subscription duration based on the selected plan
        duration_months = get_subscription_duration(gender, plan_id)
        # Maximum end date for the subscription (February 28, 2025)
        max_end_date = datetime(2025, 2, 28)

        roll = random.random()

        if roll < 0.40:  # 40% chance to end in April
            preferred_end_months = [4]
        elif roll < 0.65:  # 25% chance to end in December
            preferred_end_months = [12]
        elif roll < 0.72:  # 7% chance to end in February
            preferred_end_months = [2]
        elif roll < 0.79:  # 7% chance to end in June
            preferred_end_months = [6]
        elif roll < 0.84:  # 7% chance to end in September
            preferred_end_months = [9]
        elif roll < 0.87:  # 7% chance to end in November
            preferred_end_months = [11]
        elif roll < 0.93:  # 1% chance to end in January
            preferred_end_months = [1]
        elif roll < 0.94:  # 1% chance to end in March
            preferred_end_months = [3]
        elif roll < 0.96:  # 1% chance to end in May
           preferred_end_months = [5]
        elif roll < 0.97:  # 1% chance to end in July
           preferred_end_months = [7]
        elif roll < 0.98:  # 1% chance to end in August
           preferred_end_months = [8]
        else:  # 1% chance to end in October
           preferred_end_months = [10]

        # Add up to 12 months to match the preferred end month
        for extra_months in range(12):
           potential_end_date            =            start_date            +
relativedelta(months=duration_months + extra_months)
           if potential_end_date.month in preferred_end_months: # Check if the
potential end month matches the preferred month
               duration_months += extra_months # Adjust the duration to match
the preferred end month
               break
```

```
        # Ensure that the end date does not exceed the maximum allowed date
(max_end_date)
        max_duration_months = (max_end_date.year - start_date.year) * 12 +
(max_end_date.month - start_date.month)
        duration_months = min(duration_months, max_duration_months) # Ensure
duration doesn't exceed the allowed months

        # Calculate subscription end date based on plan duration
        end_date = start_date + relativedelta(months=duration_months)

        # Ensure end_date does not exceed max_end_date
        if end_date > max_end_date:
            end_date = max_end_date

        # Probability of having a null end_date based on plan_id
        null_end_prob = {1: 0.27, 2: 0.48, 3: 0.73}
        if random.random() < null_end_prob[plan_id]:
            end_date = None

        # Ensure end_date does not exceed max_end_date
        if end_date is not None:
            end_date = min(end_date, max_end_date)

        # If end_date is valid or the start_date is earlier than end_date,
break the loop
        if end_date is None or start_date < end_date:
            break

    # Append subscription data to the list
    subscriptions.append({
        "subscription_id": subscription_id,
        "user_id": uid,
        "plan_id": plan_id,
        "start_date": start_date.date(),
        "end_date": end_date.date() if end_date else None
    })

# Process each subscription in the list
for sub in subscriptions:
    sid = sub["subscription_id"]
    uid = sub["user_id"]

    # Convert start_date to datetime format
    start_date = datetime.combine(sub["start_date"], datetime.min.time())
```

```
    # Convert end_date to datetime format if it's not None
    end_date = datetime.combine(sub["end_date"], datetime.min.time()) if
sub["end_date"] else None

    # Retrieve user information based on user_id
    user_info = next(u for u in users if u["user_id"] == uid)

    # Extract user age
    age = user_info["age"]

    # Determine payment methods and their respective weights based on age
    if age < 35:
        payment_methods = [1,2,3,4,5,6,7]
        payment_weights = [10,6,3,7,42,28,4]
    else:
        payment_methods = [1,2,3,4,5,6,7]
        payment_weights = [37,13,4,20,15,6,5]

    # Initialise the first payment date
    pay_date = start_date
    while True:
        # If the payment date reaches or exceeds the end date, break the loop
        if end_date and pay_date >= end_date:
            break
        # If there's no end date and the payment date exceeds February 28,
2025, break the loop
        if not end_date and pay_date >= datetime(2025, 2, 28):
            break
        # Append the payment details
        payments.append({
            "payment_id": generate_payment_id(),
            "user_id": uid,
            "subscription_id": sid,
            "payment_method_id":              random.choices(payment_methods,
weights=payment_weights, k=1)[0],
            "payment_date": pay_date.date()
        })
        # Move the payment date to the next month
        pay_date += relativedelta(months=1)

# Generate a set of unique available release dates for the issues
available_dates = set()
while len(available_dates) < num_issues:
    # Add randomly generated dates between the start and end date ranges
```

```
    available_dates.add(fake.date_between(start_date=start_date_range,
end_date=end_date_range))
# Sort the available dates in ascending order
available_dates = sorted(available_dates)

# Assign an issue ID and release date to each issue
for i_num, r_date in enumerate(available_dates, start=1):
    iid = generate_issue_id() # Generate a unique issue ID
    issues.append({
        "issue_id": iid,
        "issue_number": i_num, # Assign issue number sequentially
        "release_date": r_date # Assign release date
    })

# Assign topics to each issue
for iss in issues:
    iid = iss["issue_id"]
    assigned = set() # Set to store unique assigned topics
    n_topics = random.randint(1,3) # Randomly assign 1 to 3 topics
    while len(assigned) < n_topics:
        topic_id = random.randint(1,15) # Select a random topic ID from 1 to
15
        assigned.add(topic_id) # Add topic to the set of assigned topics
    for t in assigned:
        issue_topics.append({"issue_id": iid, "topic_id": t}) # Associate each
issue with its assigned topics

# Favorite topics for different demographic groups
young_female_fav_topics = [1,3,5,7,14]
young_male_fav_topics = [1,6,12,14,15]
young_nonbinary_fav_topics = [1,6,7,14,15]
mid_female_fav_topics = [1,4,7,9,11]
mid_male_fav_topics = [1,4,8,11,12]
mid_nonbinary_fav_topics = [1,4,7,11,15]
old_female_fav_topics = [1,4,7,10,11]
old_male_fav_topics = [1,2,9,10,12]
old_nonbinary_fav_topics = [1,3,9,10,15]

# Initialise a dictionary to store user favorite topics
user_fav_topics = {}
# Loop through each user to assign their favorite topics based on age and
gender
for u in users:
    uid = u["user_id"]
    age = u["age"]
```

```
    g = u["gender"]

    # Assign favorite topics for users aged 18 to 24
    if 18 <= age <= 24:
        if g == "female":
            user_fav_topics[uid] = young_female_fav_topics # Favorite topics
for young female users
        elif g == "male":
            user_fav_topics[uid] = young_male_fav_topics # Favorite topics for
young male users
        else:
            user_fav_topics[uid]  =  young_nonbinary_fav_topics  #  Favorite
topics for young nonbinary users
    # Assign favorite topics for users aged 25 to 34
    elif 25 <= age <= 34:
        if g == "female":
            user_fav_topics[uid] = mid_female_fav_topics # Favorite topics for
middle-aged female users
        elif g == "male":
            user_fav_topics[uid] = mid_male_fav_topics # Favorite topics for
middle-aged male users
        else:
            user_fav_topics[uid] = mid_nonbinary_fav_topics # Favorite topics
for middle-aged nonbinary users
    # Assign favorite topics for users aged 35 and above
    else:
        if g == "female":
            user_fav_topics[uid] = old_female_fav_topics # Favorite topics for
older female users
        elif g == "male":
            user_fav_topics[uid] = old_male_fav_topics # Favorite topics for
older male users
        else:
            user_fav_topics[uid] = old_nonbinary_fav_topics # Favorite topics
for older nonbinary users

# Create a mapping of user_id to their respective plan_id from subscriptions
user_plan_map = {sub["user_id"]: sub["plan_id"] for sub in subscriptions}
# Initialise a mapping of user_id to an empty set of read issues
user_issue_map = {u["user_id"]: set() for u in users}

# Loop through each subscription to assign issues based on user's subscription
plan and favorite topics
for sub in subscriptions:
    uid = sub["user_id"]
```

```
    sub_start = datetime.combine(sub["start_date"], datetime.min.time())
    sub_end = (datetime.combine(sub["end_date"], datetime.min.time())
               if sub["end_date"] else datetime(2025, 2, 28))
    plan_id = user_plan_map[uid]
    fav_topics = user_fav_topics[uid]

    # List to store weighted issues based on the user's favorite topics
    weighted_issues = []
    for i in issues:
        iid = i["issue_id"]
        r_date = i["release_date"]
        # Skip issues outside the subscription period
        if not (sub_start.date() <= r_date <= sub_end.date()):
            continue
        # Get the topic IDs for the current issue
        issue_topic_ids = [row["topic_id"] for row in issue_topics if
row["issue_id"] == iid]
        # Calculate the weight for each issue based on the number of matching
favorite topics
        w = sum(1 for t in issue_topic_ids if t in fav_topics)
        # Add the issue to the weighted list if it matches favorite topics
        if w > 0:
            weighted_issues.extend([i] * (w * 5))  # Add the issue multiple
times based on its weight

    # If no weighted issues found, use all issues as valid
    valid_issues = weighted_issues if weighted_issues else issues
    read_issues = user_issue_map[uid] # Get the set of issues already read by
the user

    # Determine the number of issues the user will read based on their
subscription plan
    if plan_id == 3:
        num_read = random.randint(3,7)
    elif plan_id == 2:
        num_read = random.randint(2,5)
    else:
        num_read = random.randint(1,2)

    # Loop to select and assign issues for the user
    for _ in range(num_read):
        # Filter out issues that the user has already read
        avail = [i for i in valid_issues if i["issue_id"] not in read_issues]
        if not avail:
            break
```

```
        chosen = random.choice(avail)
        iid = chosen["issue_id"]
        read_issues.add(iid)

        # Calculate the valid reading period for the chosen issue
        latest_start = max(chosen["release_date"], sub_start.date()) # Start
date cannot be before the issue release date
        earliest_end = sub_end.date() # End date cannot be after the
subscription end date
        if latest_start > earliest_end: # If the issue's valid period is outside
the subscription period, skip
            continue
        # Generate a random reading start date within the valid period
        start_date = fake.date_between(start_date=latest_start,
end_date=earliest_end)
        # Get the user's age for further processing
        user_age = next(u["age"] for u in users if u["user_id"] == uid)

        # Determine the user's age group based on their age
        if 18 <= user_age <= 24:
            age_group = "young"
        elif 25 <= user_age <= 34:
            age_group = "mid"
        else:
            age_group = "old"

        # Get the gender of the user based on their user_id
        gender = next(u["gender"] for u in users if u["user_id"] == uid)

        # Calculate the reading probability and percentage based on the
subscription plan, gender, and age group
        if plan_id == 3:
            if gender == "female":
                base_prob = 0.95 if age_group == "young" else (0.85 if
age_group == "mid" else 0.75)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(85.00, 99.99),2)
            elif gender == "male":
                base_prob = 0.30 if age_group == "young" else (0.20 if
age_group == "mid" else 0.10)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(20.00, 80.00),2)
            else:
                base_prob = 0.50 if age_group == "young" else (0.40 if
age_group == "mid" else 0.30)
```

```
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(40.00, 95.00),2)
        elif plan_id == 2:
            if gender == "female":
                base_prob = 0.85 if age_group == "young" else (0.75 if
age_group == "mid" else 0.65)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(50.00, 99.99),2)
            elif gender == "male":
                base_prob = 0.20 if age_group == "young" else (0.15 if
age_group == "mid" else 0.10)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(10.00, 70.00),2)
            else:
                base_prob = 0.40 if age_group == "young" else (0.30 if
age_group == "mid" else 0.20)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(30.00, 75.00),2)
        else:
            if gender == "female":
                base_prob = 0.70 if age_group == "young" else (0.60 if
age_group == "mid" else 0.50)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(70.00, 99.99),2)
            elif gender == "male":
                base_prob = 0.10 if age_group == "young" else (0.07 if
age_group == "mid" else 0.05)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(5.00, 50.00),2)
            else:
                base_prob = 0.25 if age_group == "young" else (0.20 if
age_group == "mid" else 0.15)
                perc_read = 100.00 if random.random() < base_prob else
round(random.uniform(20.00, 75.00),2)

        # Append the reading session details for the user
        reading_sessions.append({
            "session_id": generate_session_id(),
            "user_id": uid,
            "issue_id": iid,
            "start_date": start_date,
            "end_date": sub["end_date"] if sub["end_date"] else None,
            "percentage_read": perc_read
        })
```

```
# Convert lists into DataFrames and save them as CSV files
users_df = pd.DataFrame(users)
subscriptions_df = pd.DataFrame(subscriptions)
payments_df = pd.DataFrame(payments)
issues_df = pd.DataFrame(issues)
issue_topics_df = pd.DataFrame(issue_topics)
reading_sessions_df = pd.DataFrame(reading_sessions)

# Save each DataFrame as a CSV file
users_df.to_csv("users.csv", index=False)
subscriptions_df.to_csv("subscriptions.csv", index=False)
payments_df.to_csv("payments.csv", index=False)
issues_df.to_csv("issues.csv", index=False)
issue_topics_df.to_csv("issue_topics.csv", index=False)
reading_sessions_df.to_csv("user_reading_sessions.csv", index=False)

print("Data generation completed!")
```

7.4 Appendix D: Business Insights Code

Appendix D relates to section 5 of the report and provides the SQLite code used for extracting business insights from the database. Step 1 is for user distribution, step 2 is for preferred payment methods, step 3 considers revenue, step 4 gives churn rate, step 5 is for content popularity, and step 6 is for user engagement.

Step 1. User Distribution

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Connect to SQLite database
conn = sqlite3.connect('magazine.db')

# SQL Query to count users per age group, gender, and subscription type
query = """
SELECT
    CASE
        WHEN age BETWEEN 18 AND 24 THEN '18-24'
        WHEN age BETWEEN 25 AND 34 THEN '25-34'
        WHEN age BETWEEN 35 AND 49 THEN '35-49'
        ELSE '50+'
    END AS age_group,
    gender,
    sp.plan_name AS subscription_type,
    COUNT(u.user_id) AS user_count
FROM users u
LEFT JOIN subscriptions s ON u.user_id = s.user_id
LEFT JOIN subscription_plans sp ON s.plan_id = sp.plan_id
GROUP BY age_group, gender, sp.plan_name
ORDER BY age_group, gender, sp.plan_name;
"""

# Execute SQL query and load results into a Pandas DataFrame
users_statistics_df = pd.read_sql_query(query, conn)

# Close the database connection
conn.close()

# Display the first few rows of the DataFrame
users_statistics_df.head()
```

Step 2. Preferred Payment Methods

```
import sqlite3
import pandas as pd

# Connect to the SQLite database
conn = sqlite3.connect('magazine.db')

# Execute SQL query
query = """
SELECT
    u.age,
    u.gender,
    sp.plan_name AS subscription_type,
    pm.payment_method_type,
    COUNT(p.payment_id) AS payment_count
FROM payments p
JOIN users u ON p.user_id = u.user_id
JOIN subscriptions s ON p.subscription_id = s.subscription_id
JOIN subscription_plans sp ON s.plan_id = sp.plan_id
JOIN payment_methods pm ON p.payment_method_id = pm.payment_method_id
GROUP BY u.age, u.gender, sp.plan_name, pm.payment_method_type
ORDER BY payment_count DESC;
"""

# Convert SQL query result into a Pandas DataFrame
Preferred_Payment_method_df = pd.read_sql_query(query, conn)

# Close the database connection
conn.close()

# Display the first few rows of the DataFrame
Preferred_Payment_method_df.head()
```

Step 3. Revenue

```
import sqlite3
import pandas as pd

# Connect to SQLite database
conn = sqlite3.connect('magazine.db')

# Updated SQL Query
query = """
WITH user_revenue AS (
    SELECT
        p.user_id,
        u.age,
        u.gender,
        sp.plan_name AS subscription_type,
        COUNT(p.payment_id) AS duration_month,  -- Number of payments =
Subscription duration in months
        SUM(CAST(REPLACE(sp.plan_price, '£', '') AS REAL)) AS
total_revenue  -- Ensure numeric conversion
    FROM payments p
    JOIN users u ON p.user_id = u.user_id
    JOIN subscriptions s ON p.subscription_id = s.subscription_id
    JOIN subscription_plans sp ON s.plan_id = sp.plan_id
    GROUP BY p.user_id, u.age, u.gender, sp.plan_name
)

SELECT
    age,
    gender,
    subscription_type,
    AVG(duration_month) AS avg_duration_month,  -- Average subscription
duration in months
    AVG(total_revenue) AS avg_revenue_per_user,
    SUM(total_revenue) AS total_revenue
FROM user_revenue
GROUP BY age, gender, subscription_type
ORDER BY age, gender, subscription_type;
"""

# Execute SQL query and load results into a Pandas DataFrame
Revenue_per_User_df = pd.read_sql_query(query, conn)

# Close the database connection
conn.close()
```

```
# Display the first few rows of the DataFrame
Revenue_per_User_df.head()
```

Step 4. Churn Rate

4.1 Converting empty end_date to null

```
import sqlite3

conn = sqlite3.connect('magazine.db')
cursor = conn.cursor()

# Define and execute the UPDATE query, Converting empty end date to null
update_query = """
UPDATE subscriptions
SET end_date = NULL
WHERE end_date = '';
"""
cursor.execute(update_query)

conn.commit()

cursor.close()
conn.close()

print("Successfully updated empty-string end_date to NULL.")
```

4.2 Churn Rate by age group, gender, and subscription type

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('magazine.db')

# Define the query to calculate churn rate based on age group, gender, and
subscription type
churn_query = """
WITH subscription_info AS (
    SELECT
        CASE
            WHEN u.age BETWEEN 18 AND 24 THEN '18-24'
            WHEN u.age BETWEEN 25 AND 34 THEN '25-34'
            WHEN u.age BETWEEN 35 AND 49 THEN '35-49'
            WHEN u.age >= 50 THEN '50+'
            ELSE 'Other'
        END AS age_group,
        u.gender,
        sp.plan_name AS subscription_type,
        COUNT(*) AS total_subscriptions,
        SUM(CASE WHEN s.end_date IS NOT NULL THEN 1 ELSE 0 END) AS
ended_subscriptions
    FROM subscriptions s
    JOIN users u ON s.user_id = u.user_id
    JOIN subscription_plans sp ON s.plan_id = sp.plan_id
    GROUP BY age_group, u.gender, sp.plan_name
)
SELECT
    age_group,
    gender,
    subscription_type,
    total_subscriptions,
    ended_subscriptions,
    ROUND(
        (CAST(ended_subscriptions AS FLOAT) / total_subscriptions) * 100,
        2
    ) AS churn_rate_percentage
FROM subscription_info
ORDER BY age_group, gender, subscription_type;
"""
churn_df = pd.read_sql_query(churn_query, conn)
conn.close()

# Data Processing
```

```
gender_order = ["female", "male", "non-binary", "prefer not to say"]
churn_df["gender"] = pd.Categorical(churn_df["gender"],
categories=gender_order, ordered=True)
plan_order = ["Basic", "Standard", "Premium"]
churn_df["subscription_type"] = pd.Categorical(churn_df["subscription_type"],
categories=plan_order, ordered=True)

# Display the first few rows of the DataFrame
churn_df.head()
```

4.3 Churn Rate Seasonality

```
import sqlite3
import pandas as pd
import calendar

conn = sqlite3.connect("magazine.db")

# Calculate the average number of ended subscriptions per month
monthly_avg_query = """
WITH monthly_ends AS (
    SELECT
        strftime('%Y-%m', end_date) AS year_month,
        CAST(strftime('%m', end_date) AS INT) AS month_num,
        COUNT(*) AS ended_subscriptions
    FROM subscriptions
    WHERE end_date IS NOT NULL
    GROUP BY year_month
)
SELECT
    month_num,
    AVG(ended_subscriptions) AS avg_ended
FROM monthly_ends
GROUP BY month_num
ORDER BY month_num;
"""

season_df = pd.read_sql_query(monthly_avg_query, conn)
conn.close()

# Convert month_num to numeric (handle any potential issues)
season_df['month_num'] = pd.to_numeric(season_df['month_num'],
errors='coerce')

# Drop rows where month_num is NaN
season_df = season_df.dropna(subset=['month_num'])

# Ensure month_num is an integer and filter valid months (1-12)
season_df['month_num'] = season_df['month_num'].astype(int)
season_df = season_df[season_df['month_num'].between(1, 12)]

# Map month numbers to abbreviated month names (e.g., 1 -> Jan, 2 -> Feb)
season_df['month_name'] = season_df['month_num'].apply(lambda m:
calendar.month_abbr[m])

# Round the average ended subscriptions to two decimal places
```

```
season_df['avg_ended'] = season_df['avg_ended'].round(2)

# Display the first few rows of the DataFrame
season_df.head()
```

Step 5. Content Popularity

5.1 Topic Popularity by Age Group 18-24

```
import sqlite3
import pandas as pd

conn = sqlite3.connect("magazine.db")

#Topic Popularity by Age Group 18-24
df_top_topics_18_24 = pd.read_sql_query('''
    WITH RankedTopics AS (
        SELECT
            '18-24' AS age_group,
            u.gender,
            t.topic_name AS topic,
            COUNT(*) AS popularity_count,
            RANK() OVER (PARTITION BY u.gender ORDER BY COUNT(*) DESC) AS
topic_rank
        FROM user_reading_sessions urs
        JOIN users u ON urs.user_id = u.user_id
        JOIN issues i ON urs.issue_id = i.issue_id
        JOIN issue_topics it ON i.issue_id = it.issue_id
        JOIN topics t ON it.topic_id = t.topic_id
        WHERE u.gender IN ('male', 'female', 'non-binary') AND u.age BETWEEN
18 AND 24
        GROUP BY u.gender, t.topic_name
    )
    SELECT age_group, gender, topic, popularity_count
    FROM RankedTopics
    WHERE topic_rank <= 5
    ORDER BY gender, topic_rank
''', conn)

conn.close()

# Display the first few rows of the DataFrame
df_top_topics_18_24.head()
```

5.2 Topic Popularity by Age Group 25-34

```
import sqlite3
import pandas as pd

conn = sqlite3.connect("magazine.db")

df_top_topics_25_34 = pd.read_sql_query('''
    WITH RankedTopics AS (
        SELECT
            '25-34' AS age_group,
            u.gender,
            t.topic_name AS topic,
            COUNT(*) AS popularity_count,
            RANK() OVER (PARTITION BY u.gender ORDER BY COUNT(*) DESC) AS
topic_rank
        FROM user_reading_sessions urs
        JOIN users u ON urs.user_id = u.user_id
        JOIN issues i ON urs.issue_id = i.issue_id
        JOIN issue_topics it ON i.issue_id = it.issue_id
        JOIN topics t ON it.topic_id = t.topic_id
        WHERE u.gender IN ('male', 'female', 'non-binary') AND u.age BETWEEN
25 AND 34
        GROUP BY u.gender, t.topic_name
    )
    SELECT age_group, gender, topic, popularity_count
    FROM RankedTopics
    WHERE topic_rank <= 5
    ORDER BY gender, topic_rank
''', conn)

conn.close()


# Data Preprocessing

# Identify the top 5 topics overall
top_5_topics =
df_top_topics_25_34.groupby("topic")["popularity_count"].sum().nlargest(5).in
dex

# Filter DataFrame to include only those top 5 topics
df_top_topics_25_34_filtered =
df_top_topics_25_34[df_top_topics_25_34["topic"].isin(top_5_topics)]

# Display the first few rows of the DataFrame
```

563111, 5662419, 5665831, 5667011, 5667395, 5673899

```
df_top_topics_25_34.head()
```

5.3 Topic Popularity by Age Group 35-49

```
import sqlite3
import pandas as pd

conn = sqlite3.connect("magazine.db")

df_top_topics_35_49 = pd.read_sql_query('''
    WITH RankedTopics AS (
        SELECT
            '35-49' AS age_group,
            u.gender,
            t.topic_name AS topic,
            COUNT(*) AS popularity_count,
            RANK() OVER (PARTITION BY u.gender ORDER BY COUNT(*) DESC) AS
topic_rank
        FROM user_reading_sessions urs
        JOIN users u ON urs.user_id = u.user_id
        JOIN issues i ON urs.issue_id = i.issue_id
        JOIN issue_topics it ON i.issue_id = it.issue_id
        JOIN topics t ON it.topic_id = t.topic_id
        WHERE u.gender IN ('male', 'female', 'non-binary') AND u.age BETWEEN
35 AND 49
        GROUP BY u.gender, t.topic_name
    )
    SELECT age_group, gender, topic, popularity_count
    FROM RankedTopics
    WHERE topic_rank <= 5
    ORDER BY gender, topic_rank
''', conn)

conn.close()

# Data Preprocessing

# Identify the top 5 topics overall
top_5_topics =
df_top_topics_35_49.groupby("topic")["popularity_count"].sum().nlargest(5).in
dex

# Filter DataFrame to include only those top 5 topics
df_top_topics_35_49_filtered =
df_top_topics_35_49[df_top_topics_35_49["topic"].isin(top_5_topics)]

# Display the first few rows of the DataFrame
df_top_topics_35_49.head()
```

Step 6. User Engagement

6.1 Top Readers

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('magazine.db')

# Define the SQL query to analyse reading statistics and identify top users
query = """
WITH ReadingStats AS (
    SELECT
        urs.user_id,
        s.plan_id,
        COUNT(urs.session_id) AS total_reading_sessions,
        SUM(CASE WHEN urs.percentage_read = 100 THEN 1 ELSE 0 END) AS
completed_reading_sessions
    FROM user_reading_sessions urs
    JOIN subscriptions s ON urs.user_id = s.user_id
    GROUP BY urs.user_id, s.plan_id
),
RankedUsers AS (
    SELECT
        rs.user_id,
        sp.plan_name AS subscription_type,
        rs.total_reading_sessions,
        rs.completed_reading_sessions,
        RANK() OVER (PARTITION BY rs.plan_id ORDER BY
rs.total_reading_sessions DESC) AS read_rank,
        RANK() OVER (PARTITION BY rs.plan_id ORDER BY
rs.completed_reading_sessions DESC) AS completion_rank
    FROM ReadingStats rs
    JOIN subscription_plans sp ON rs.plan_id = sp.plan_id
)
SELECT
    subscription_type,
    user_id AS most_reading_user,
    total_reading_sessions AS most_read_sessions,
    user_id AS most_completion_user,
    completed_reading_sessions AS most_completed_sessions
FROM RankedUsers
WHERE read_rank = 1 OR completion_rank = 1
ORDER BY subscription_type;
"""
```

```
Reading_Stats_df = pd.read_sql_query(query, conn)
conn.close()

# Display the first few rows of the DataFrame
Reading_Stats_df.head()
```

6.2 Analysis of User Reading Patterns by Age, Gender, and Subscription Plan

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('magazine.db')

# Define the SQL query to analyse overall user reading patterns
query = """
SELECT
    u.age,
    u.gender,
    sp.plan_name AS subscription_type,
    urs.percentage_read
FROM user_reading_sessions urs
JOIN users u ON urs.user_id = u.user_id
JOIN subscriptions s ON u.user_id = s.user_id
JOIN subscription_plans sp ON s.plan_id = sp.plan_id;
"""
reading_df = pd.read_sql_query(query, conn)
conn.close()

# Data Preprocessing
# Define age groups
reading_df['age_group'] = pd.cut(
    reading_df['age'],
    bins=[18, 25, 35, 50, 100],
    labels=['18-24', '25-34', '35-49', '50+'],
    ordered=True
)

# Define subscription type order
subscription_order = ['Basic', 'Standard', 'Premium']
reading_df['subscription_type'] = pd.Categorical(
    reading_df['subscription_type'],
    categories=subscription_order,
    ordered=True
)

# Display the first few rows of the DataFrame
reading_df.head()
```

6.3 Users Reading More than 85 Percent by Subscription Plan

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('magazine.db')

# Define SQL query to analyse users reading more than 85% by Subscription
Plan
df_reading_above_85 = pd.read_sql_query('''
    SELECT
        sp.plan_name AS subscription_plan,
        COUNT(urs.user_id) AS high_reading_count
    FROM user_reading_sessions urs
    JOIN users u ON urs.user_id = u.user_id
    JOIN subscriptions s ON u.user_id = s.user_id
    JOIN subscription_plans sp ON s.plan_id = sp.plan_id
    WHERE urs.percentage_read > 85
    GROUP BY sp.plan_name
''', conn)
conn.close()

# Ensure correct order of subscription plans
order = ["Basic", "Standard", "Premium"]
df_reading_above_85 =
df_reading_above_85.set_index("subscription_plan").loc[order].reset_index()

# Display the first few rows of the DataFrame
df_reading_above_85.head()
```

7.5 Appendix E: Business Insights Visualizations Code

Appendix E relates to section 5 of the report and provides the SQLite code used for visualizing business insights from the database. These visualizations match up with the insights code from Appendix F. Step 1 is for user distribution, step 2 is for preferred payment methods, step 3 considers revenue, step 4 gives churn rate, step 5 is for content popularity, and step 6 is for user engagement.

Step 1. User Distribution

1.1 Overall Statistics

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Data Prerprocessing for visualisation
age_group_counts =
users_statistics_df.groupby('age_group')['user_count'].sum()
gender_counts = users_statistics_df.groupby('gender')['user_count'].sum()
subscription_counts =
users_statistics_df.groupby('subscription_type')['user_count'].sum()

subscription_order = ['Basic', 'Standard', 'Premium']
subscription_counts = subscription_counts.reindex(subscription_order)

# -------------------------
# 1. Define "BABA" Theme
# -------------------------
baba_palette = [
    "#002F5D",  # Deep navy
    "#00509D",  # Medium/darker blue
    "#3B86D1",  # Brighter blue
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)


# -------------------------
# 2. Helper Function
# -------------------------
```

```
def add_value_labels(ax):
    """Add value labels above bars with extra padding for visibility."""
    for bar in ax.patches:
        val = bar.get_height()
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            val + (0.05 * ax.get_ylim()[1]),  # Adds extra padding
dynamically
            f"{int(val)}",
            ha="center",
            va="bottom",
            color=baba_palette[0],  # Dark navy text
            fontsize=12,
            fontweight="bold"
        )

# -------------------------
# 3. Plot: Users by Age Group
# -------------------------
plt.figure(figsize=(10, 5), facecolor="white")
ax = sns.barplot(
    x=age_group_counts.index,
    y=age_group_counts.values,
    hue=age_group_counts.index,
    palette=[baba_palette[2]] * len(age_group_counts)  # Brighter blue
)
plt.title("Number of Users by Age Group", color=baba_palette[0],
fontweight="bold", fontsize=16, pad=20)
plt.xlabel("Age Group", color=baba_palette[0], fontsize=18)
plt.ylabel("User Count", color=baba_palette[0], fontsize=18)
plt.ylim(0, max(age_group_counts.values) * 1.15)  # Expands y-limit
dynamically
add_value_labels(ax)
plt.xticks(color=baba_palette[0], fontsize=12)
plt.yticks(color=baba_palette[0], fontsize=12)
plt.tight_layout()
plt.show()

# -------------------------
# 4. Plot: Users by Gender
# -------------------------
plt.figure(figsize=(10, 5), facecolor="white")
ax = sns.barplot(
    x=gender_counts.index,
    y=gender_counts.values,
```

```
    hue=gender_counts.index,
    palette=[baba_palette[1]] * len(gender_counts)  # Medium blue
)
plt.title("Number of Users by Gender", color=baba_palette[0],
fontweight="bold", fontsize=16, pad=20)
plt.xlabel("Gender", color=baba_palette[0], fontsize=18)
plt.ylabel("User Count", color=baba_palette[0], fontsize=18)
plt.ylim(0, max(gender_counts.values) * 1.15)
add_value_labels(ax)
plt.xticks(color=baba_palette[0], fontsize=12)
plt.yticks(color=baba_palette[0], fontsize=12)
plt.tight_layout()
plt.show()

# -------------------------
# 5. Plot: Users by Subscription Type
# -------------------------
plt.figure(figsize=(10, 5), facecolor="white")
ax = sns.barplot(
    x=subscription_counts.index,
    y=subscription_counts.values,
    hue=subscription_counts.index,
    palette=baba_palette[:3]  # Different shades of blue
)
plt.title("Number of Users by Subscription Type", color=baba_palette[0],
fontweight="bold", fontsize=16, pad=20)
plt.xlabel("Subscription Type", color=baba_palette[0], fontsize=18)
plt.ylabel("User Count", color=baba_palette[0], fontsize=18)
plt.ylim(0, max(subscription_counts.values) * 1.15)
add_value_labels(ax)
plt.xticks(color=baba_palette[0], fontsize=12)
plt.yticks(color=baba_palette[0], fontsize=12)
plt.tight_layout()
plt.show()
```

Step 2. Preferred Payment Methods

2.1 Overall Preferred Payment Methods

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme with Matched Colors
# -------------------------
payment_palette = {
    "apple_pay": "#002F5D",      # Deep Navy
    "bank_transfer": "#00509D",  # Dark Blue
    "credit_card": "#3B86D1",    # Brighter Blue
    "debit_card": "#F28E2B",     # Soft Orange
    "gift_card": "#4F6D7A",      # Muted Blue (Replaced Red)
    "google_pay": "#A843CC",     # Purple
    "paypal": "#FFC20A"          # Yellow-Gold
}

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation
# -------------------------
# Summing the total payment counts by payment method
payment_summary = Preferred_Payment_method_df.groupby(
    'payment_method_type', observed=False  # Avoid FutureWarning
)['payment_count'].sum().reset_index()

# Sorting payment methods by total usage in descending order
payment_summary = payment_summary.sort_values(by='payment_count',
ascending=False)

# Display the summarized table
```

```
display(payment_summary)

# -------------------------
# 3. Helper Function
# -------------------------
def add_value_labels(ax):
    """Add value labels above bars with extra padding for visibility."""
    for bar in ax.patches:
        val = bar.get_width()
        ax.text(
            val + (0.02 * ax.get_xlim()[1]),  # Adds extra padding
dynamically
            bar.get_y() + bar.get_height() / 2,
            f"{int(val)}",
            ha="left",
            va="center",
            color="#002F5D",  # Dark navy text
            fontsize=12,
            fontweight="bold"
        )

# -------------------------
# 4. Visualisation: Most Frequently Used Payment Methods
# -------------------------
plt.figure(figsize=(10, 6), facecolor="white")
ax = sns.barplot(
    x="payment_count",
    y="payment_method_type",
    data=payment_summary,
    palette=payment_palette  # Uses exact colors from the provided plot
)

plt.title("Most Frequently Used Payment Methods", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Total Payment Count", fontsize=18, color="#002F5D")
plt.ylabel("Payment Method", fontsize=18, color="#002F5D")
plt.xlim(0, max(payment_summary["payment_count"]) * 1.15)  # Expands x-axis
limit dynamically

# Add labels to bars
add_value_labels(ax)

plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)
plt.tight_layout()
```

```
plt.show()
```

2.2 Payment Method Usage by Age Group

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme + Distinct Colors
# -------------------------
baba_palette = [
    "#002F5D",  # Deep Navy (Primary)
    "#00509D",  # Dark Blue
    "#3B86D1",  # Bright Blue
    "#F4A259",  # Warm Orange (contrast)
    "#D62839",  # Deep Red (contrast)
    "#9B5DE5",  # Purple (for differentiation)
    "#FCCB00",  # Gold/Yellow (for clear contrast)
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation (Updated Age Groups)
# -------------------------
Preferred_Payment_method_df['age_group'] = pd.cut(
    Preferred_Payment_method_df['age'],
    bins=[18, 25, 35, 50, 100],  # Only using 18-24, 25-34, 35-49, 50+
    labels=['18-24', '25-34', '35-49', '50+'],
    ordered=True  # Ensuring correct order
)

# Summarizing payment methods by age group
age_payment_summary = Preferred_Payment_method_df.groupby(
    ['age_group', 'payment_method_type'], observed=False
)['payment_count'].sum().unstack()
```

```
# Displaying the table in Colab
display(age_payment_summary)

# -------------------------
# 3. Visualisation: Payment Methods by Age Group
# -------------------------
fig, ax = plt.subplots(figsize=(12, 6), facecolor="white")

# Stacked Bar Plot
age_payment_summary.plot(
    kind="bar",
    stacked=True,
    color=baba_palette[:len(age_payment_summary.columns)],  # Uses only
needed colors
    ax=ax
)

# -------------------------
# 4. Formatting and Readability
# -------------------------
plt.title("Payment Method Usage by Age Group", fontsize=16,
fontweight="bold", color=baba_palette[0], pad=20)
plt.xlabel("Age Group", fontsize=18, color=baba_palette[0])
plt.ylabel("Total Payment Count", fontsize=18, color=baba_palette[0])
plt.xticks(rotation=0, color=baba_palette[0], fontsize=12)
plt.yticks(color=baba_palette[0], fontsize=12)

# Adjust legend placement for better readability
ax.legend(title="Payment Method", bbox_to_anchor=(1.05, 1), loc="upper left",
frameon=False)

plt.tight_layout()
plt.show()
```

Step 3. Revenue

3.1 Average Subscription Duration by Gender

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme + Lighter Pastel Colors
# -------------------------
baba_pastel_palette = [
    "#74B3CE",  # Light Blue
    "#A3D9FF",  # Soft Sky Blue
    "#FFC857",  # Warm Yellow
    "#FF9B85",  # Soft Coral
    "#C085F7",  # Light Purple
    "#A5FFD6",  # Mint Green
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation
# -------------------------
# Summarizing average subscription duration by gender
gender_duration_summary = Revenue_per_User_df.groupby(
    ['gender', 'subscription_type'], observed=False
)['avg_duration_month'].mean().unstack()

# Display the summarized table
display(gender_duration_summary)

# -------------------------
# 3. Visualisation: Subscription Duration by Gender
# -------------------------
```

```
fig, ax = plt.subplots(figsize=(12, 6), facecolor="white")

# Stacked Bar Plot
gender_duration_summary.plot(
    kind="bar",
    stacked=True,
    color=baba_pastel_palette[:len(gender_duration_summary.columns)],  # Uses
pastel colors
    ax=ax
)

# -------------------------
# 4. Formatting for Readability
# -------------------------
plt.title("Average Subscription Duration by Gender", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Gender", fontsize=14, color="#002F5D")
plt.ylabel("Average Duration (Months)", fontsize=14, color="#002F5D")
plt.xticks(rotation=0, color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

# Adjust legend placement for better readability
ax.legend(title="Subscription Plan", bbox_to_anchor=(1.05, 1), loc="upper
left", frameon=False)

plt.tight_layout()
plt.show()
```

3.2 Average Subscription Duration by Subscription Plan

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme with Softer Colors
# -------------------------
baba_pastel_palette = [
    "#74B3CE",  # Light Blue
    "#A3D9FF",  # Soft Sky Blue
    "#FFC857",  # Warm Yellow
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation (Remove Gender Split)
# -------------------------
# Define subscription type order
subscription_order = ['Basic', 'Standard', 'Premium']

# Convert 'subscription_type' to categorical with specific order
Revenue_per_User_df['subscription_type'] = pd.Categorical(
    Revenue_per_User_df['subscription_type'],
    categories=subscription_order,
    ordered=True
)

# Summarising average subscription duration by **subscription type only**
plan_duration_summary = Revenue_per_User_df.groupby(
    'subscription_type', observed=False
)['avg_duration_month'].mean().reset_index()

# Reorder rows based on subscription type order
```

```
plan_duration_summary =
plan_duration_summary.set_index('subscription_type').loc[subscription_order]

# Display the table
display(plan_duration_summary)

# -------------------------
# 3. Visualisation: Average Subscription Duration by Plan
# -------------------------
fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")

# Bar Plot (without gender split)
sns.barplot(
    x=plan_duration_summary.index,
    y=plan_duration_summary['avg_duration_month'],
    palette=baba_pastel_palette,
    ax=ax
)

# -------------------------
# 4. Formatting & Readability
# -------------------------
plt.title("Average Subscription Duration by Subscription Plan", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Subscription Plan", fontsize=14, color="#002F5D")
plt.ylabel("Average Duration (Months)", fontsize=14, color="#002F5D")
plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

# Adjust y-axis limit to prevent text cutoff
max_value = plan_duration_summary['avg_duration_month'].max()
plt.ylim(0, max_value + 5)  # Adds extra space at the top

# Add values on bars with adjusted positioning
for bar in ax.patches:
    height = bar.get_height()
    ax.text(
        bar.get_x() + bar.get_width() / 2,
        height + 2,  # Moves text slightly higher
        f"{height:.1f}",
        ha="center",
        va="bottom",
        fontsize=12,
        fontweight="bold",
        color="#002F5D"
```

```
    )

plt.tight_layout()
plt.show()
```

3.3 Total Revenue by Gender

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme Colors (Bluish with Complementary Shades)
# -------------------------
baba_palette = [
    "#74B3CE",  # Light Blue
    "#A3D9FF",  # Soft Sky Blue
    "#FFD166",  # Gold-Yellow (soft contrast)
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation
# -------------------------
# Summarising total revenue by gender and subscription type
gender_revenue_summary = Revenue_per_User_df.groupby(
    ['gender', 'subscription_type'], observed=False
)['total_revenue'].sum().unstack()

# -------------------------
# 3. Visualisation: Total Revenue by Gender
# -------------------------
fig, ax = plt.subplots(figsize=(12, 6), facecolor="white")

# Stacked Bar Plot with Improved Bluish Theme
gender_revenue_summary.plot(
    kind="bar",
    stacked=True,
    color=baba_palette[:len(gender_revenue_summary.columns)],  # Using soft
blues + complementary color
```

```
    ax=ax
)

# -------------------------
# 4. Formatting & Readability
# -------------------------
plt.title("Total Revenue by Gender", fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel("Gender", fontsize=14, color="#002F5D")
plt.ylabel("Total Revenue (£)", fontsize=14, color="#002F5D")
plt.xticks(rotation=0, color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

# Adjust legend for better readability
ax.legend(title="Subscription Plan", bbox_to_anchor=(1.05, 1), loc="upper
left", frameon=False)

plt.tight_layout()
plt.show()
```

3.4 Total Revenue by Subscription Plan

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import warnings

# Suppress warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# -------------------------
# 1. Define "BABA" Theme Colors (Better Contrast & Readability)
# -------------------------
baba_palette = [
    "#5A9EC9",  # Muted Blue (Primary)
    "#B4D4FF",  # Soft Blue (Lighter Accent)
    "#FFCF56",  # Gold (Warm Contrast)
    "#8DDCA4",  # Light Green (Cool Contrast)
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preparation (Summarising revenue by subscription type)
# -------------------------
plan_revenue_summary = Revenue_per_User_df.groupby(
    ['subscription_type', 'gender'], observed=False
)['total_revenue'].sum().unstack()

# Display the table
#display(plan_revenue_summary)

# -------------------------
# 3. Visualisation: Total Revenue by Subscription Plan
# -------------------------
fig, ax = plt.subplots(figsize=(12, 6), facecolor="white")

# Stacked Bar Plot with Improved Readability
plan_revenue_summary.plot(
    kind="bar",
```

```
    stacked=True,
    color=baba_palette[:len(plan_revenue_summary.columns)],  # Using muted
blues + complementary gold & green
    ax=ax
)

# -------------------------
# 4. Formatting & Readability
# -------------------------
plt.title("Total Revenue by Subscription Plan", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Subscription Plan", fontsize=18, color="#002F5D")
plt.ylabel("Total Revenue (£)", fontsize=18, color="#002F5D")
plt.xticks(rotation=0, color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

# Adjust legend for better readability
ax.legend(title="Gender", bbox_to_anchor=(1.05, 1), loc="upper left",
frameon=False)

plt.tight_layout()
plt.show()
```

Step 4. Churn Rate

4.1 Churn rate based on age group, gender, and subscription type

```
# -------------------------
# 1. Define "BABA" Theme Colors (Cool & Warm Balance)
# -------------------------
baba_palette = [
    "#5A9EC9",  # Muted Blue (Primary)
    "#A3D9FF",  # Soft Sky Blue (Lighter Accent)
    "#FFC857",  # Gold (Warm Contrast)
    "#8DDCA4",  # Light Green (Cool Contrast)
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Helper Function: Add Value Labels
# -------------------------
def add_value_labels(ax):
    for bar in ax.patches:
        height = bar.get_height()
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            height + 1,  # Padding for visibility
            f"{height:.1f}%",
            ha="center",
            va="bottom",
            fontsize=12,
            fontweight="bold",
            color="#002F5D"
        )

# -------------------------
# 3. Visualisation: Churn by Age Group
# -------------------------
age_churn = churn_df.groupby('age_group',
as_index=False)[['ended_subscriptions', 'total_subscriptions']].sum()
age_churn['churn_rate'] = (age_churn['ended_subscriptions'] /
age_churn['total_subscriptions']) * 100
```

```
fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")
bars = sns.barplot(
    x=age_churn['age_group'],
    y=age_churn['churn_rate'],
    palette=baba_palette[:4],  # Apply color theme
    ax=ax
)

plt.title('Churn Rate by Age Group', fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel('Age Group', fontsize=14, color="#002F5D")
plt.ylabel('Churn Rate (%)', fontsize=14, color="#002F5D")
plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

add_value_labels(ax)
plt.tight_layout()
plt.show()

# -------------------------
# 4. Visualisation: Churn by Gender
# -------------------------
gender_churn = churn_df.groupby('gender',
as_index=False)[['ended_subscriptions', 'total_subscriptions']].sum()
gender_churn['churn_rate'] = (gender_churn['ended_subscriptions'] /
gender_churn['total_subscriptions']) * 100

fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")
bars = sns.barplot(
    x=gender_churn['gender'],
    y=gender_churn['churn_rate'],
    palette=[baba_palette[0], baba_palette[1], baba_palette[2],
baba_palette[3]],  # Better contrast
    ax=ax
)

plt.title('Churn Rate by Gender', fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel('Gender', fontsize=14, color="#002F5D")
plt.ylabel('Churn Rate (%)', fontsize=14, color="#002F5D")
plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

add_value_labels(ax)
plt.tight_layout()
```

```
plt.show()

# -------------------------
# 5. Visualisation: Churn by Subscription Plan
# -------------------------
plan_churn = churn_df.groupby('subscription_type',
as_index=False)[['ended_subscriptions', 'total_subscriptions']].sum()
plan_churn['churn_rate'] = (plan_churn['ended_subscriptions'] /
plan_churn['total_subscriptions']) * 100

fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")
bars = sns.barplot(
    x=plan_churn['subscription_type'],
    y=plan_churn['churn_rate'],
    palette=[baba_palette[2], baba_palette[0], baba_palette[3]],  # Rotated
for distinction
    ax=ax
)

plt.title('Churn Rate by Subscription Plan', fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel('Subscription Plan', fontsize=18, color="#002F5D")
plt.ylabel('Churn Rate (%)', fontsize=18, color="#002F5D")
plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

add_value_labels(ax)
plt.tight_layout()
plt.show()
```

4.2 Churn Rate Seasonality

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import calendar
import seaborn as sns
import warnings

warnings.filterwarnings("ignore", category=UserWarning)


# -------------------------
# 1. Define "BABA" Theme Colors
# -------------------------
baba_palette = [
    "#74B3CE",  # Light Blue
    "#A3D9FF",  # Soft Sky Blue
    "#00509D",  # Deep Blue for Contrast
]

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 4. Visualization
# -------------------------
fig, ax = plt.subplots(figsize=(9, 5), facecolor="white")

bars = sns.barplot(
    x=season_df['month_name'],
    y=season_df['avg_ended'],
    palette=baba_palette,
    ax=ax
)

# -------------------------
# 5. Formatting & Readability
# -------------------------
plt.title("Average Ended Subscriptions by Month (All Years)", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Month", fontsize=14, color="#002F5D")
```

```
plt.ylabel("Average Number of Ended Subscriptions", fontsize=14,
color="#002F5D")
plt.xticks(color="#002F5D", fontsize=12)
plt.yticks(color="#002F5D", fontsize=12)

# Adjust top margin to prevent cut-offs
plt.ylim(0, max(season_df['avg_ended']) * 1.15 if not season_df.empty else 1)

# Add values on bars
for bar in bars.patches:
    height = bar.get_height()
    ax.text(
        bar.get_x() + bar.get_width()/2,
        height + 0.2,  # Adjust padding
        f"{height:.2f}",
        ha="center",
        va="bottom",
        fontsize=12,
        fontweight="bold",
        color="#002F5D"
    )

plt.subplots_adjust(top=0.85)
plt.show()
```

Step 5. Content Popularity

5.1 Topic Popularity by Age Group 18-24

```
# -------------------------
# 1. Define "BABA" Theme Colors
# -------------------------
baba_palette = {
    "female": "#74B3CE",  # Light Blue
    "male": "#00509D",  # Deep Blue
    "non-binary": "#F4A259",  # Warm Orange for Contrast
}

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Data Preprocessing
# -------------------------
df_pivot = df_top_topics_18_24.pivot(index="topic", columns="gender",
values="popularity_count").fillna(0)

# -------------------------
# 3. Visualisation
# -------------------------
fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")

# Plot stacked bars
df_pivot.plot(
    kind="bar",
    stacked=True,
    ax=ax,
    color=[baba_palette.get(col, "#CCCCCC") for col in df_pivot.columns]  #
Ensures assigned colors
)

# -------------------------
# 5. Formatting & Readability
# -------------------------
plt.title("Top Topics for 18-24 Age Group Across Gender", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Topic", fontsize=14, color="#002F5D")
```

```
plt.ylabel("Popularity Count", fontsize=14, color="#002F5D")
plt.xticks(rotation=45, ha="right", fontsize=12, color="#002F5D")  # Improved
label rotation
plt.yticks(fontsize=12, color="#002F5D")

# Adjust legend position
ax.legend(title="Gender", bbox_to_anchor=(1.05, 1), loc="upper left")

plt.subplots_adjust(bottom=0.25, top=0.85)  # Ensure labels fit well
plt.show()
```

5.2 Topic Popularity by Age Group 25-34

```
# -------------------------
# 1. Define "BABA" Theme Colors
# -------------------------
baba_palette = {
    "female": "#74B3CE",  # Light Blue
    "male": "#00509D",  # Deep Blue
    "non-binary": "#F4A259",  # Warm Orange for Contrast
}

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Visualisation
# -------------------------
# Pivot for stacked bar chart
df_pivot = df_top_topics_25_34_filtered.pivot(index="topic",
columns="gender", values="popularity_count").fillna(0)

fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")

# Plot stacked bars
df_pivot.plot(
    kind="bar",
    stacked=True,
    ax=ax,
    color=[baba_palette.get(col, "#CCCCCC") for col in df_pivot.columns]  #
Ensures assigned colors
)

# -------------------------
# 3. Formatting & Readability
# -------------------------
plt.title("Top 5 Topics for 25-34 Age Group", fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel("Topic", fontsize=14, color="#002F5D")
plt.ylabel("Popularity Count", fontsize=14, color="#002F5D")
plt.xticks(rotation=45, ha="right", fontsize=12, color="#002F5D")  # Improved
label rotation
plt.yticks(fontsize=12, color="#002F5D")
```

```
# Adjust legend position
ax.legend(title="Gender", bbox_to_anchor=(1.05, 1), loc="upper left")

plt.subplots_adjust(bottom=0.25, top=0.85)  # Ensure labels fit well
plt.show()
```

5.3 Topic Popularity by Age Group 35-49

```
# ------------------------
# 1. Define "BABA" Theme Colors
# ------------------------
baba_palette = {
    "female": "#74B3CE",  # Light Blue
    "male": "#00509D",  # Deep Blue
    "non-binary": "#F4A259",  # Warm Orange for Contrast
}

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# ------------------------
# 2. Visualisation
# ------------------------

# Pivot for stacked bar chart
df_pivot = df_top_topics_35_49_filtered.pivot(index="topic",
columns="gender", values="popularity_count").fillna(0)

fig, ax = plt.subplots(figsize=(10, 6), facecolor="white")

# Plot stacked bars
df_pivot.plot(
    kind="bar",
    stacked=True,
    ax=ax,
    color=[baba_palette.get(col, "#CCCCCC") for col in df_pivot.columns]  #
Ensures assigned colors
)

# ------------------------
# 5. Formatting & Readability
# ------------------------
plt.title("Top 5 Topics for 35-49 Age Group", fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel("Topic", fontsize=14, color="#002F5D")
plt.ylabel("Popularity Count", fontsize=14, color="#002F5D")
plt.xticks(rotation=45, ha="right", fontsize=12, color="#002F5D")  # Improved
label rotation
```

```
plt.yticks(fontsize=12, color="#002F5D")

# Adjust legend position
ax.legend(title="Gender", bbox_to_anchor=(1.05, 1), loc="upper left")

plt.subplots_adjust(bottom=0.25, top=0.85)  # Ensure labels fit well
plt.show()
```

Step 6. User Engagement

6.1 Analysis of User Reading Patterns by Age, Gender, and Subscription Plan

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# -------------------------
# 1. Define "BABA" Theme Colors
# -------------------------
baba_palette = {
    "age": "#74B3CE",  # Light Blue
    "gender": "#00509D",  # Deep Blue
    "subscription": "#4A90E2",  # Medium Blue (Replaced Orange)
}

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)

# -------------------------
# 2. Helper Function for Labels
# -------------------------
def add_value_labels(ax):
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(
            f'{height:.1f}%',
            (p.get_x() + p.get_width() / 2., height + 2),
            ha='center', va='bottom',
            fontsize=12, fontweight='bold', color="#002F5D"
        )

# -------------------------
# 3. Visualisation: Avg % Read by Age Group
# -------------------------
plt.figure(figsize=(10,5))
age_means = reading_df.groupby('age_group')['percentage_read'].mean()
ax = sns.barplot(x=age_means.index, y=age_means.values,
color=baba_palette["age"])
```

```
plt.title("Average % Read by Age Group", fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel("Age Group", fontsize=14, color="#002F5D")
plt.ylabel("Average % Read", fontsize=14, color="#002F5D")
plt.xticks(fontsize=12, color="#002F5D")
plt.yticks(fontsize=12, color="#002F5D")

plt.ylim(0, 110)
add_value_labels(ax)
plt.tight_layout()
plt.show()

# -------------------------
# 4. Visualisation: Avg % Read by Gender
# -------------------------
plt.figure(figsize=(10,5))
gender_means = reading_df.groupby('gender')['percentage_read'].mean()
ax = sns.barplot(x=gender_means.index, y=gender_means.values,
color=baba_palette["gender"])

plt.title("Average % Read by Gender", fontsize=16, fontweight="bold",
color="#002F5D", pad=20)
plt.xlabel("Gender", fontsize=14, color="#002F5D")
plt.ylabel("Average % Read", fontsize=14, color="#002F5D")
plt.xticks(fontsize=12, color="#002F5D")
plt.yticks(fontsize=12, color="#002F5D")

plt.ylim(0, 110)
add_value_labels(ax)
plt.tight_layout()
plt.show()

# -------------------------
# 5. Visualisation: Avg % Read by Subscription Type
# -------------------------
plt.figure(figsize=(10,5))
plan_means =
reading_df.groupby('subscription_type')['percentage_read'].mean()
ax = sns.barplot(x=plan_means.index, y=plan_means.values,
color=baba_palette["subscription"])

plt.title("Average % Read by Subscription Type", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Subscription Type", fontsize=18, color="#002F5D")
plt.ylabel("Average % Read", fontsize=18, color="#002F5D")
```

```
plt.xticks(fontsize=12, color="#002F5D")
plt.yticks(fontsize=12, color="#002F5D")

plt.ylim(0, 110)
add_value_labels(ax)
plt.tight_layout()
plt.show()
```

6.2 Users Reading More Than 85 Percent by Subscription Plan

```
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# -------------------------
# 1. Define "BABA" Theme Colors (Replaced Orange with Blue)
# -------------------------
baba_palette = ["#74B3CE", "#00509D", "#4A90E2"]  # Light blue, Deep blue,
Medium blue

sns.set_style("whitegrid", {
    "axes.facecolor": "white",
    "axes.edgecolor": "#E2E2E2",
    "grid.color": "#E2E2E2"
})
sns.set_context("talk", font_scale=1.1)


# -------------------------
# 3. Visualisation
# -------------------------
fig, ax = plt.subplots(figsize=(9, 6), facecolor="white")

# Bar Plot
sns.barplot(
    x=df_reading_above_85["subscription_plan"],
    y=df_reading_above_85["high_reading_count"],
    palette=baba_palette,  # Uses new blue color palette
    ax=ax
)

# -------------------------
# 4. Formatting & Readability
# -------------------------
plt.title("Users Reading More Than 85% by Subscription Plan", fontsize=16,
fontweight="bold", color="#002F5D", pad=20)
plt.xlabel("Subscription Plan", fontsize=18, color="#002F5D")
plt.ylabel("Users with >85% Reading Percentage", fontsize=18,
color="#002F5D")
plt.xticks(fontsize=12, color="#002F5D")
plt.yticks(fontsize=12, color="#002F5D")

# Add values on bars
```

```
for bar in ax.patches:
    height = bar.get_height()
    ax.text(
        bar.get_x() + bar.get_width() / 2,
        height + 5,  # Extra padding for visibility
        f"{int(height)}",
        ha="center",
        va="bottom",
        fontsize=12,
        fontweight="bold",
        color="#002F5D"
    )

plt.grid(axis="y", linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```