Localization Data Intern Test

1 - Description of the dataset

This dataset contains information about a hypothetical free-to-play mobile game on the year 2016. They are three tables named ***account***, ***account_date_session*** and ***iap_purchase***.

1.1 – account

Describe the ***account*** table:

- **account_id**: Unique identifier for user profiles.
- **created_time**: Timestamp of account creation.
- **created_device**: Device used during account creation.
- **created_platform**: Platform (e.g., iOS, Android) of account creation.
- **country_code**: Country code of the user's location.
- **created_app_store_id**: Identifier for the app store used for account creation.

1.2 – account_date_session

Describe the ***account_date_session*** table:

- **account_id**: Links to the **account** table for user identification.
- **date**: Date of activity.
- **session_count**: Number of sessions for a given date.
- **session_duration_sec**: Duration of sessions in seconds.

1.3 – iap_purchase

Describe the **iap_purchase** table:

- **account_id**: Relates to the **account** table for user identification.
- **created_time**: Timestamp of in-app purchase.
- **package_id_hash**: Hashed identifier for the purchased package.
- **iap_price_usd_cents**: In-app purchase price in USD cents.
- **app_store_id**: Identifier for the app store where the purchase occurred.

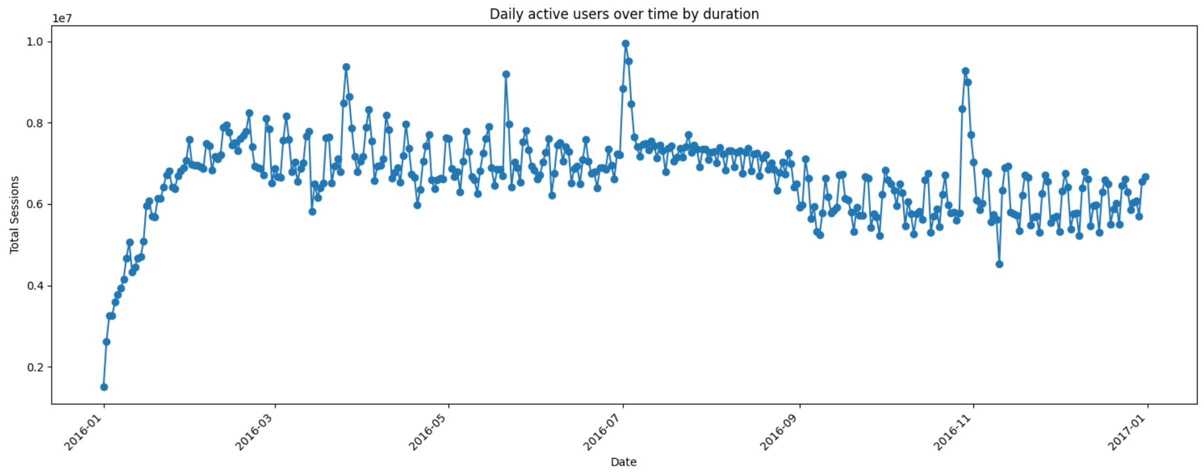
2 – Analyse the daily active users:

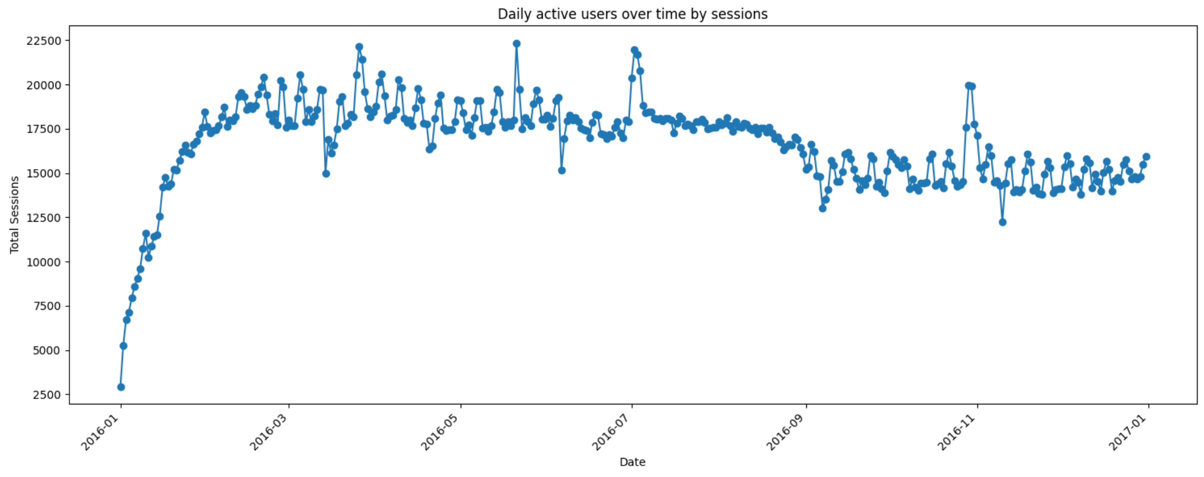
To analyse trends in data and active users we need to get all sessions for all users and group them by date. By doing that we can plot a graph showing us ups and drops.

Query used:

```
SELECT date, SUM(session_count) as total_sessions, SUM(session_duration_sec)
as total_duration FROM account_date_session GROUP BY date;
```

Now we can plot it:



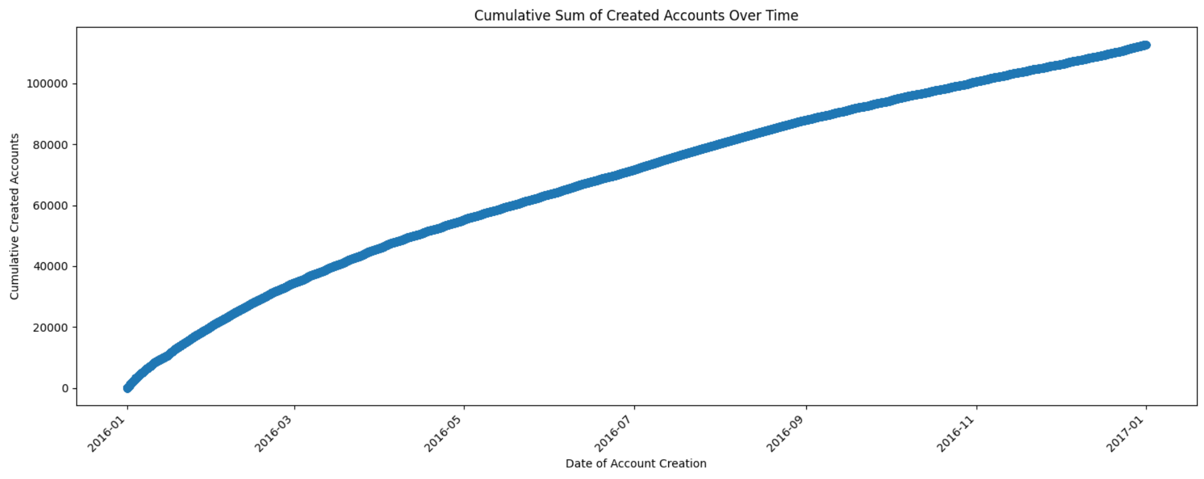


We can see that total sessions and total duration are likely the same. In both charts we can say that the game has developed a lot, mainly in the first 3 months, probably because the game just released and went hype.

They kept a strong base even if the number of sessions decreased a bit. Each peak is probably an update in the game and the fact that total sessions decreased mainly between July and October is probably because of no update.

For small drops we can suppose maybe bugs in the game causing less sessions from users.

We can also display a chart to see the cumulative sum of accounts created over time:

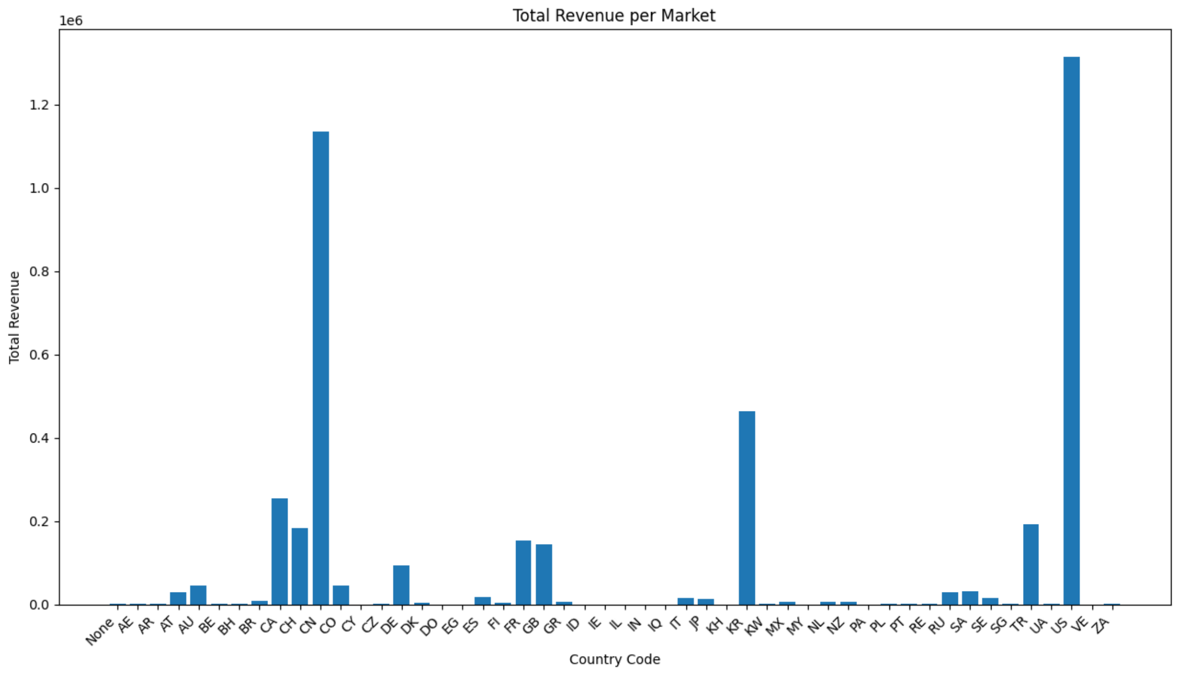


As shown by this curve, it is a very linear progression, reaching 112 792 accounts at the end of 2016.

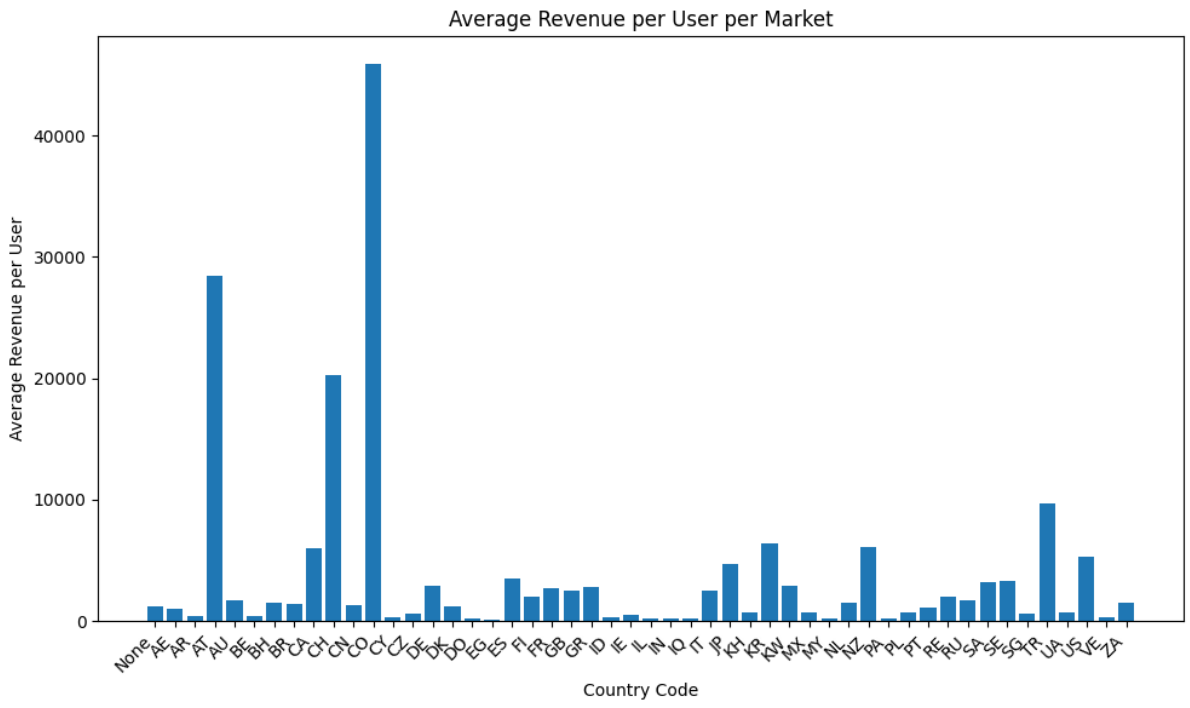
3 – Analyse sales:

For sales analyses, we can use iap_purchase table and calculate the sum for each country, then divided the result by the amount of user for each country to get the average revenue per user per country.

Here below are two charts, the first one with the total and the second one with the average revenue per user per country:

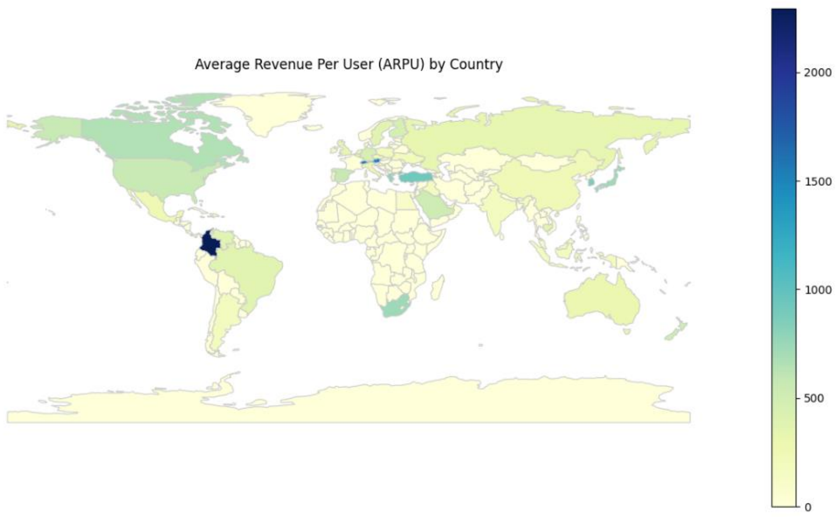


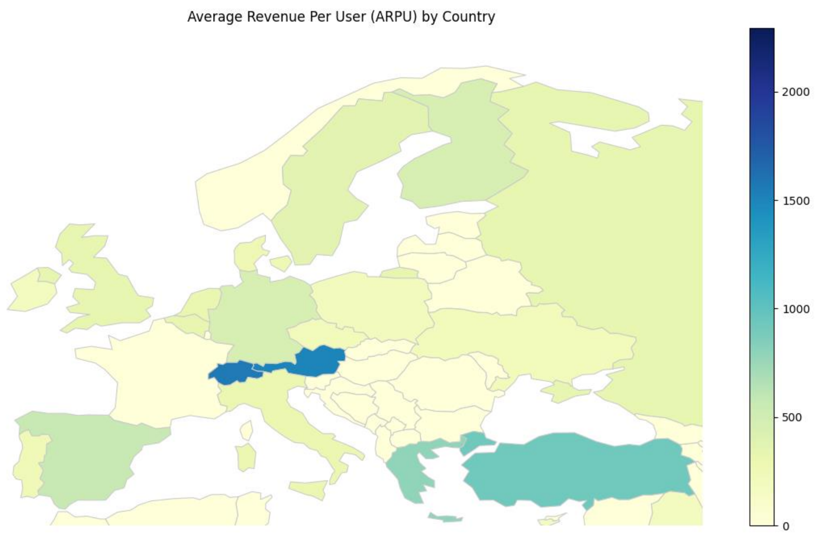
For the total we can see that the top three are United States, China, and Korea. These are countries with many users.



Things changes when you represent average revenue per user. Now the top three are Colombia, Austria, and Switzerland. These are countries with very few users (only one for Austria and Colombia, 9 for Switzerland) but he paid a big amount.

Another representation can be with a map instead:





To summarize the sales part, we can say that most of in-app purchases are based in a few big countries where people are used to play mobile games and pay.

It could be a good idea to try to improve the market in places where revenue per user is high, because where are not so many users but it seems that they pay in-app purchases more.

Conclusion

This mobile game got off to a great start, with a huge surge in users and updates to keep players active. As far as in-app purchases are concerned, this remains mainly in a few countries, but there seem to be promising markets in some others.

Code written to do the test.

```python
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import geopandas as gpd
import pycountry

# Connect to the SQLite database
conn = sqlite3.connect('sample.sqlite')

# Helper function to execute a query and return a DataFrame
def query_to_dataframe(query):
    return pd.read_sql_query(query, conn)

# Helper function to display table description in a pretty table
def display_table_description(table_name, description_df):
    table = PrettyTable()
    table.field_names = ["Column Name", "Type", "Nullable"]

    for _, row in description_df.iterrows():
        table.add_row([row['name'], row['type'], "YES" if row['notnull'] == 0
else "NO"])

    print(f"Description of {table_name} table:")
    print(table)

def convert_iso_alpha2_to_alpha3(alpha2_code):
    try:
        country = pycountry.countries.get(alpha_2=alpha2_code)
        return country.alpha_3 if country else None
    except LookupError:
        return None

def convert_iso_alpha2_to_country(alpha2_code):
    try:
        country = pycountry.countries.get(alpha_2=alpha2_code)
        return country.name if country else None
    except LookupError:
        return None

# Question 1: Give a short description of datasets
query1 = "PRAGMA table_info(account);"
description_account = query_to_dataframe(query1)
display_table_description("account", description_account)

query2 = "PRAGMA table_info(account_date_session);"
```

```
description_account_date_session = query_to_dataframe(query2)
display_table_description("account_date_session",
description_account_date_session)

query3 = "PRAGMA table_info(iap_purchase);"
description_iap_purchase = query_to_dataframe(query3)
display_table_description("iap_purchase", description_iap_purchase)

print("Description of account table:")
print(description_account)
print("\nDescription of account_date_session table:")
print(description_account_date_session)
print("\nDescription of iap_purchase table:")
print(description_iap_purchase)

# Question 2: Analyze the daily active users
# Compare DAU changes over time
query_dau = "SELECT date, SUM(session_count) as total_sessions,
SUM(session_duration_sec) as total_duration FROM account_date_session GROUP BY
date;"
dau_data = query_to_dataframe(query_dau)
dau_data['date'] = pd.to_datetime(dau_data['date'])
dau_data.set_index('date', inplace=True)

# Plot DAU changes over time by session
plt.figure(figsize=(15, 6))
plt.plot(dau_data.index, dau_data['total_sessions'], marker='o', linestyle='-
')
plt.title('Daily active users over time by sessions')
plt.xlabel('Date')
plt.ylabel('Total Sessions')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
readability
plt.tight_layout()
plt.show()


# Plot DAU changes over time by duration
plt.figure(figsize=(15, 6))
plt.plot(dau_data.index, dau_data['total_duration'], marker='o', linestyle='-
')
plt.title('Daily active users over time by duration')
plt.xlabel('Date')
plt.ylabel('Total Sessions')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
readability
plt.tight_layout()
plt.show()
```

```python
# New Question: Visualize the Cumulative Sum of Created Accounts Over Time
query_created_accounts = "SELECT created_time, COUNT(account_id) as
created_accounts FROM account GROUP BY created_time;"
created_accounts_data = query_to_dataframe(query_created_accounts)
created_accounts_data['created_time'] =
pd.to_datetime(created_accounts_data['created_time'])

# Calculate the cumulative sum of created accounts
created_accounts_data['cumulative_created_accounts'] =
created_accounts_data['created_accounts'].cumsum()

# Plot the cumulative sum of created accounts over time
plt.figure(figsize=(15, 6))
plt.plot(created_accounts_data['created_time'],
created_accounts_data['cumulative_created_accounts'], marker='o', linestyle='-
')
plt.title('Cumulative Sum of Created Accounts Over Time')
plt.xlabel('Date of Account Creation')
plt.ylabel('Cumulative Created Accounts')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
readability
plt.tight_layout()
plt.show()

print("\nDescription of daily active users over time:")
print(created_accounts_data)

# Question 3: Analyze sales
# Analyze the geographic split of revenue and users
query_geo = "SELECT a.country_code, SUM(i.iap_price_usd_cents) as
total_revenue, COUNT(DISTINCT i.account_id) as total_users FROM iap_purchase i
LEFT JOIN account a ON i.account_id = a.account_id GROUP BY a.country_code;"
geo_data = query_to_dataframe(query_geo)

# Calculate average revenue per user per market
geo_data['avg_revenue_per_user'] = geo_data['total_revenue'] /
geo_data['total_users']
geo_data['country'] =
geo_data['country_code'].apply(convert_iso_alpha2_to_country)
# Display the geographic split of revenue and users
print("\nGeographic Split of Revenue and Users:")
print(geo_data.sort_values(by=['avg_revenue_per_user']))

# Plot average revenue per user per market
plt.figure(figsize=(10, 6))
plt.bar(geo_data['country_code'].astype(str),
geo_data['avg_revenue_per_user'])
plt.title('Average Revenue per User per Market')
plt.xlabel('Country Code')
plt.ylabel('Average Revenue per User')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
readability
plt.tight_layout()
plt.show()

# Plot revenue per market
plt.figure(figsize=(10, 6))
plt.bar(geo_data['country_code'].astype(str), geo_data['total_revenue'])
plt.title('Total Revenue per Market')
plt.xlabel('Country Code')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better
readability
plt.tight_layout()
plt.show()


# Analyze the geographic split of revenue and users
query_geo = "SELECT a.country_code, AVG(i.iap_price_usd_cents) as
avg_revenue_per_user FROM iap_purchase i LEFT JOIN account a ON i.account_id =
a.account_id GROUP BY a.country_code;"
geo_data = query_to_dataframe(query_geo)
geo_data['country_code_iso3'] =
geo_data['country_code'].apply(convert_iso_alpha2_to_alpha3)

# Load the world map shapefile
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

# Merge the world map with the ARPU data
merged_data = world.merge(geo_data, left_on='iso_a3',
right_on='country_code_iso3', how='left')
merged_data['avg_revenue_per_user'] =
merged_data['avg_revenue_per_user'].fillna(0)

# Plot the choropleth map
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
merged_data.plot(column='avg_revenue_per_user', cmap='YlGnBu', linewidth=0.8,
ax=ax, edgecolor='0.8', legend=True)
ax.set_title('Average Revenue Per User (ARPU) by Country')
ax.set_axis_off()  # Turn off the axis for better visualization
plt.show()

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