DETAILED PROJECT REPORT

Table of Contents

Column Names :

1. **ID**:
   - Primary column, uniquely identifying each transaction entry.
2. **Amount**:
   - Total transaction amount associated with each ID, representing the value processed.
3. **Amount Details**:

- Comprises the base transaction amount along with the Stripe fee charged.

4. **Card**:
   - Specifies the type of card used for the transaction (e.g., Visa, Mastercard).
5. **Cardholder**:
   - Unique identifier for the cardholder associated with the transaction.
6. **Customer City & Country**:
   - Location (city and country) where the transaction occurred.
7. **Created**:
   - Timestamp indicating the date and time the transaction took place, recorded in UNIX format.
8. **Currency**:
   - Currency in which the transaction was processed.
9. **Type**:
   - Specifies the nature of the transaction, such as "charged" or "refund."
10. **Fee**:
    - Stripe fee applied to the transaction.
11. **Customer Transaction**:
    - Amount paid back to the customer out of the total transaction amount.
12. **Net Amount**:
    - Net revenue earned by the company, calculated as `Amount - Customer Transaction - Stripe Fee`.
13. **Payment Failed**:
    - Indicator of transaction status, with `0` denoting successful transactions and `1` indicating failed transactions.
14. **Payment Occurrence**:
    - Number of attempts made to successfully complete the transaction.

15. **Date**:
    - Human-readable date derived from the converted UNIX timestamp in the "Created" column.

Python script to import data from stripe

```
!pip install requests
!pip install stripe
import requests
import datetime

# Stripe API endpoint for issuing transactions
url = "https://api.stripe.com/v1/issuing/transactions"

# Set up the API key (test key for example)
api_key = "stripe_secret_key"

# Time filter: Last 30 days
thirty_days_ago = int((datetime.datetime.now() - datetime.timede

# Define the headers and parameters for the request
headers = {
    "Authorization": f"Bearer {api_key}"
}

params = {
    "limit": 100,  # Adjust as needed
    "created[gte]": thirty_days_ago
}

# Send the GET request
response = requests.get(url, headers=headers, params=params)

# Check if the request was successful
```

```
if response.status_code == 200:
    transactions = response.json()
    for transaction in transactions['data']:
        print(transaction)
else:
    print(f"Failed to fetch data: {response.status_code}, {respo
```

SQL Queries to Analyze the Data

car_rental_transactions.csv

```
USE RENTAL;

SELECT * FROM DBO.car_rental_transactions;

SELECT HOST_NAME() AS LOCAL_HOST_NAME;

--CREATED A NEW TABLE "RENTAL_COPY" WITH MODIFIED UNIX TIMESTAMP
SELECT *, DATEADD(SECOND,CREATED,'1970-01-01') AS DATE
INTO RENTAL_COPY
FROM DBO.car_rental_transactions;


select * from RENTAL_COPY;

--FILTERING THE DATA FROM 2024-10-01 TO 2024-10-31
SELECT * FROM RENTAL_COPY
WHERE DATE BETWEEN '2024-10-01' AND '2024-10-31';

--ALTERNATIVE WAY OF FILTERING LAST 30 DAYS DATA
```

```
SELECT * FROM RENTAL_COPY
WHERE DATE >= DATEADD(DAY,-30, GETDATE()) AND DATE < GETDATE();

SELECT * FROM RENTAL_COPY
WHERE DATE >= DATEADD(DAY,-30, GETDATE()) AND DATE < GETDATE()
order by DATE;

--Grouping the data week-wise
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)

SELECT Week_Number,
       MIN(Week_Start) AS Week_Start_Date,
       MAX(Week_End) AS Week_End_Date,
       COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Week_Number;

--Average Transaction Amount Week-wise
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER, AVG(AMOUNT) AS AVERAGE_TRANSACTION_AMOUNT
FROM WeeklyTransactions
```

```
GROUP BY Week_Number
ORDER BY Week_Number;

--AVERAGE TRANSACTION IN DESCENDING ORDER
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER, AVG(AMOUNT) AS AVERAGE_TRANSACTION_AMOUNT
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY AVERAGE_TRANSACTION_AMOUNT DESC;

--Average Transaction Amount VS Transaction Count
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER, AVG(AMOUNT) AS AVERAGE_TRANSACTION_AMOUNT, C
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY AVERAGE_TRANSACTION_AMOUNT DESC;

--Transaction Count with start and end date of Week
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
```

```
            DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)

SELECT Week_Number,
       MIN(Week_Start) AS Week_Start_Date,
       MAX(Week_End) AS Week_End_Date,
       COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Transaction_Count DESC;

--WEEK 2 HAS Maximum nos of Transactions
--WEEK 3 & 5 have lesser nos of Transactions

--FILTERING AMOUNT, CUSTOMER_EARNING, STRIPE_FEE  & NET_AMOUNT F
SELECT AMOUNT, (CUSTOMER_TRANSACTION) AS CUSTOMER_EARNING , MERC
FROM RENTAL_COPY;



--ANALYZING THE DATA ON WEEKLY BASIS OF TOTAL AMOUNT OF TRANSACT
--CHARGED BY STRIPE AND TOTAL PROFIT EARNED BY THE COMPANY
WITH WeeklyTransactions AS (
    SELECT *,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER,
MIN(WEEK_START) AS Week_Start_Date,
MAX(WEEK_END) AS Week_End_Date,
SUM(AMOUNT) as Total_Amount,
SUM(CUSTOMER_TRANSACTION) as Total_Customer_Earning,
```

```
SUM(FEE) as Total_Stripe_Fee,
SUM(NET_AMOUNT) as Total_Profit
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Week_Number;

--Analyzing the Data on Weekly basis for Total Amount of Transac
WITH WeeklyTransactions AS (
    SELECT DATE,
           AMOUNT,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT Week_Number,
       MIN(Week_Start) AS Week_Start_Date,
       MAX(Week_End) AS Week_End_Date,
       SUM(AMOUNT) as Total_Amount
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Total_Amount DESC;

--ANALYZING THE DATA ON WEEKLY BASIS for Total_Amount VS Transac
WITH WeeklyTransactions AS (
    SELECT DATE,
           AMOUNT,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT Week_Number,
       MIN(Week_Start) AS Week_Start_Date,
```

```
       MAX(Week_End) AS Week_End_Date,
       SUM(AMOUNT) as Total_Amount,
       COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Total_Amount DESC;

SELECT * FROM RENTAL_COPY;

----ANALYZING THE DATA ON WEEKLY BASIS for Total_Customer_Earni
WITH WeeklyTransactions AS (
    SELECT DATE,
           CUSTOMER_TRANSACTION,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER,
MIN(WEEK_START) AS Week_Start_Date,
MAX(WEEK_END) AS Week_End_Date,
SUM(CUSTOMER_TRANSACTION) as Total_Customer_Earning,
COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Total_Customer_Earning DESC;

--ANALYZING THE DATA ON WEEKLY BASIS for Total_Stripe_Fee VS Tra
WITH WeeklyTransactions AS (
    SELECT DATE,
           FEE,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
```

```
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER,
MIN(WEEK_START) AS Week_Start_Date,
MAX(WEEK_END) AS Week_End_Date,
SUM(FEE) as Total_Stripe_Fee,
COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Total_Stripe_Fee DESC;

--ANALYZING THE DATA ON WEEKLY BASIS for Total_Profit VS Transac
WITH WeeklyTransactions AS (
    SELECT DATE,
           NET_AMOUNT,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER,
MIN(WEEK_START) AS Week_Start_Date,
MAX(WEEK_END) AS Week_End_Date,
SUM(NET_AMOUNT) as Total_Profit,
COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Total_Profit DESC;

WITH WeeklyTransactions AS (
    SELECT DATE,
           NET_AMOUNT,
           DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') +
           DATEADD(DAY, -1 * (DATEPART(WEEKDAY, DATE) - 1), DATE
           DATEADD(DAY, 6 - (DATEPART(WEEKDAY, DATE) - 1), DATE)
```

```
    FROM RENTAL_COPY
    WHERE DATE >= '2024-09-25'
)
SELECT WEEK_NUMBER,
MIN(WEEK_START) AS Week_Start_Date,
MAX(WEEK_END) AS Week_End_Date,
SUM(NET_AMOUNT) as Total_Profit,
COUNT(*) AS Transaction_Count
FROM WeeklyTransactions
GROUP BY Week_Number
ORDER BY Week_Number;

--Filtering Refund Data
SELECT * FROM RENTAL_COPY
WHERE TYPE = 'REFUND';

--FILTERING DISTINCT CUSTOMER_CITY & CUSTOMER_COUNTRY
SELECT DISTINCT CUSTOMER_CITY, customer_country FROM RENTAL_COPY

--Analyzing the Data for each country with respect to Total Prof
--Total Stripe Fee
with WeeklyTransactions as (
select *,
        DATEPART(WEEK, DATE) - DATEPART(WEEK, '2024-09-25') + 1
        DATEADD(DAY,-1 * (DATEPART(WEEKDAY,DATE) -1) ,DATE) AS W
        DATEADD(DAY,6 - (DATEPART(WEEKDAY,DATE) -1) ,DATE) AS We
        FROM RENTAL_COPY
        WHERE DATE >= '2024-09-25')

SELECT
       CUSTOMER_COUNTRY,
       SUM(NET_AMOUNT) AS TOTAL_PROFIT,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_STRIPE_FEE
       FROM WeeklyTransactions
```

```
       GROUP BY CUSTOMER_COUNTRY
       ORDER BY CUSTOMER_COUNTRY;
SELECT * FROM RENTAL_COPY;

--Listing down all the columns of the Table
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'RENTAL_COPY';

--Creating a new Table with specific columns and concatenated Ci
SELECT ID, AMOUNT, (CUSTOMER_CITY + ' '+ CUSTOMER_COUNTRY) AS CI
INTO RENTAL_TABLE_1
FROM RENTAL_COPY;

DROP TABLE RENTAL_TABLE_1;

SELECT * FROM RENTAL_TABLE_1;

--Analyzing the Financial data City_Country wise
Select CITY_COUNTRY,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_FEE,
       SUM(PAYOUT) AS TOTAL_PAYOUT
FROM RENTAL_TABLE_1
GROUP BY CITY_COUNTRY
ORDER BY CITY_COUNTRY;

--Ordering according to Total Amount
Select CITY_COUNTRY,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_FEE,
       SUM(PAYOUT) AS TOTAL_PAYOUT
FROM RENTAL_TABLE_1
GROUP BY CITY_COUNTRY
```

```
ORDER BY TOTAL_AMOUNT DESC;

--Ordering according to Total_Customer_Transaction
Select CITY_COUNTRY,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_FEE,
       SUM(PAYOUT) AS TOTAL_PAYOUT
FROM RENTAL_TABLE_1
GROUP BY CITY_COUNTRY
ORDER BY TOTAL_CUSTOMER_TRANSACTION DESC;

----Ordering according to Total_Fee
Select CITY_COUNTRY,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_FEE,
       SUM(PAYOUT) AS TOTAL_PAYOUT
FROM RENTAL_TABLE_1
GROUP BY CITY_COUNTRY
ORDER BY TOTAL_FEE DESC;

--Ordering according to Total_Payout
Select CITY_COUNTRY,
       SUM(AMOUNT) AS TOTAL_AMOUNT,
       SUM(CUSTOMER_TRANSACTION) AS TOTAL_CUSTOMER_TRANSACTION,
       SUM(FEE) AS TOTAL_FEE,
       SUM(PAYOUT) AS TOTAL_PAYOUT
FROM RENTAL_TABLE_1
GROUP BY CITY_COUNTRY
ORDER BY TOTAL_PAYOUT DESC;


Select Distinct CITY_COUNTRY from RENTAL_TABLE_1;

--ANALYZING WEEKLY AVERAGE TRANSACTIONS FOR Brisbane Australia
```

```
With WeeklyTransactions as (
SELECT *,
        DATEPART(WEEK, DATE) - DATEPART( WEEK,'2024-09-25') + 1
        DATEADD(DAY, -1 * DATEPART(WEEKDAY,DATE) -1, DATE) AS We
        DATEADD(DAY, 6 - DATEPART(WEEKDAY, DATE) -1, DATE) AS We
        FROM RENTAL_COPY
        WHERE DATE >= '2024-09-25')
SELECT T1.Week_Number,
       MIN(T1.Week_Start) As Week_Start_Date,
       MAX(T1.Week_End) as Week_End_Date,
       T2.CITY_COUNTRY,
       AVG(T2.CUSTOMER_TRANSACTION) AS 'AVERAGE_TRANSACTION',
       AVG(T2.FEE) AS 'AVERAGE_FEE',
       AVG(T2.PAYOUT) AS 'AVERAGE_PAYOUT'
       FROM WeeklyTransactions AS T1
       JOIN RENTAL_TABLE_1 T2
       ON T1.DATE =  T2.DATE
       WHERE CITY_COUNTRY = 'Brisbane Australia'
       GROUP BY Week_Number, CITY_COUNTRY
       ORDER BY Week_Number;

--ANALYZING WEEKLY AVERAGE TRANSACTIONS FOR Jurong Singapore
With WeeklyTransactions as (
SELECT *,
        DATEPART(WEEK, DATE) - DATEPART( WEEK,'2024-09-25') + 1
        DATEADD(DAY, -1 * DATEPART(WEEKDAY,DATE) -1, DATE) AS We
        DATEADD(DAY, 6 - DATEPART(WEEKDAY, DATE) -1, DATE) AS We
        FROM RENTAL_COPY
        WHERE DATE >= '2024-09-25')
SELECT T1.Week_Number,
       MIN(T1.Week_Start) As Week_Start_Date,
       MAX(T1.Week_End) as Week_End_Date,
       T2.CITY_COUNTRY,
       AVG(T2.CUSTOMER_TRANSACTION) AS 'AVERAGE_TRANSACTION',
       AVG(T2.FEE) AS 'AVERAGE_FEE',
       AVG(T2.PAYOUT) AS 'AVERAGE_PAYOUT'
```

```
        FROM WeeklyTransactions AS T1
        JOIN RENTAL_TABLE_1 T2
        ON T1.DATE =  T2.DATE
        WHERE CITY_COUNTRY = 'Jurong Singapore'
        GROUP BY Week_Number, CITY_COUNTRY
        ORDER BY Week_Number;

----ANALYZING WEEKLY AVERAGE TRANSACTIONS FOR Melbourne Australi
With WeeklyTransactions as (
SELECT *,
        DATEPART(WEEK, DATE) - DATEPART( WEEK,'2024-09-25') + 1
        DATEADD(DAY, -1 * DATEPART(WEEKDAY,DATE) -1, DATE) AS We
        DATEADD(DAY, 6 - DATEPART(WEEKDAY, DATE) -1, DATE) AS We
        FROM RENTAL_COPY
        WHERE DATE >= '2024-09-25')
SELECT T1.Week_Number,
       MIN(T1.Week_Start) As Week_Start_Date,
       MAX(T1.Week_End) as Week_End_Date,
       T2.CITY_COUNTRY,
       AVG(T2.CUSTOMER_TRANSACTION) AS 'AVERAGE_TRANSACTION',
       AVG(T2.FEE) AS 'AVERAGE_FEE',
       AVG(T2.PAYOUT) AS 'AVERAGE_PAYOUT'
       FROM WeeklyTransactions AS T1
       JOIN RENTAL_TABLE_1 T2
       ON T1.DATE =  T2.DATE
       WHERE CITY_COUNTRY = 'Melbourne Australia'
       GROUP BY Week_Number, CITY_COUNTRY
       ORDER BY Week_Number;
--This also shows that there was no transactions made during wee

----ANALYZING WEEKLY AVERAGE TRANSACTIONS FOR Perth Australia
With WeeklyTransactions as (
SELECT *,
        DATEPART(WEEK, DATE) - DATEPART( WEEK,'2024-09-25') + 1
        DATEADD(DAY, -1 * DATEPART(WEEKDAY,DATE) -1, DATE) AS We
        DATEADD(DAY, 6 - DATEPART(WEEKDAY, DATE) -1, DATE) AS We
```

```
         FROM RENTAL_COPY
         WHERE DATE >= '2024-09-25')
SELECT T1.Week_Number,
       MIN(T1.Week_Start) As Week_Start_Date,
       MAX(T1.Week_End) as Week_End_Date,
       T2.CITY_COUNTRY,
       AVG(T2.CUSTOMER_TRANSACTION) AS 'AVERAGE_TRANSACTION',
       AVG(T2.FEE) AS 'AVERAGE_FEE',
       AVG(T2.PAYOUT) AS 'AVERAGE_PAYOUT'
       FROM WeeklyTransactions AS T1
       JOIN RENTAL_TABLE_1 T2
       ON T1.DATE =  T2.DATE
       WHERE CITY_COUNTRY = 'Perth Australia'
       GROUP BY Week_Number, CITY_COUNTRY
       ORDER BY Week_Number;

----ANALYZING WEEKLY AVERAGE TRANSACTIONS FOR Singapore Singapor
With WeeklyTransactions as (
SELECT *,
        DATEPART(WEEK, DATE) - DATEPART( WEEK,'2024-09-25') + 1
        DATEADD(DAY, -1 * DATEPART(WEEKDAY,DATE) -1, DATE) AS We
        DATEADD(DAY, 6 - DATEPART(WEEKDAY, DATE) -1, DATE) AS We
        FROM RENTAL_COPY
        WHERE DATE >= '2024-09-25')
SELECT T1.Week_Number,
       MIN(T1.Week_Start) As Week_Start_Date,
       MAX(T1.Week_End) as Week_End_Date,
       T2.CITY_COUNTRY,
       AVG(T2.CUSTOMER_TRANSACTION) AS 'AVERAGE_TRANSACTION',
       AVG(T2.FEE) AS 'AVERAGE_FEE',
       AVG(T2.PAYOUT) AS 'AVERAGE_PAYOUT'
       FROM WeeklyTransactions AS T1
       JOIN RENTAL_TABLE_1 T2
       ON T1.DATE =  T2.DATE
       WHERE CITY_COUNTRY = 'Singapore Singapore'
```

```
        GROUP BY Week_Number, CITY_COUNTRY
        ORDER BY Week_Number;
```

Converting SQL Queries to CSV File (Python Automation)

1. Converting single query to csv file

```
!pip install plyer

import pyodbc
import pandas as pd
import os
from datetime import datetime
from plyer import notification

#create SQL connection
connection = pyodbc.connect( driver = '{ODBC Driver 17 for SQL S
                            trusted_connection = 'yes')

#Read the data
sql_query = "SELECT * FROM RENTAL_COPY WHERE DATE BETWEEN '2024-

#Data from SQL to Pandas
df = pd.read_sql(sql = sql_query, con = connection)

#Export the data to the Location
df.to_csv(os.environ["userprofile"] + "\\DESKTOP\\Converted_Data
          datetime.now().strftime('%y-%m-%d %H%M%S') + ".csv" ,

#Notify when the conversion has happened
notification.notify(title = "Report Status", message = f"""Data
Total Rows : {df.shape[0]} \n Total Columns : {df.shape[1]}""",
```

2. Converting multiple queries to a single csv file , each query will be on diff sheet within single csv file

```
import pyodbc
import pandas as pd
import os
from datetime import datetime
from plyer import notification

# Create SQL connection
connection = pyodbc.connect(driver='{ODBC Driver 17 for SQL Serv

# Define your SQL queries
queries = {"DATA_1st Oct_31st Oct" : "SELECT * FROM RENTAL_COPY
            "New_modified_date_column" : "select * from RENTAL_
            "last_one_month_data" : "SELECT * FROM RENTAL_COPY

# Create a new Excel writer object
excel_filename = os.path.join(os.environ["userprofile"], "DESKTO

with pd.ExcelWriter(excel_filename) as writer:
    for sheet_name, sql_query in queries.items():
        # Read the data for each query
        df = pd.read_sql(sql=sql_query, con=connection)

        # Write each DataFrame to a separate sheet in the Excel
        df.to_excel(writer, sheet_name=sheet_name, index=False)

# Notify when the conversion has happened
notification.notify(title="Report Status", message=f"""Data has
Total Queries Processed: {len(queries)}""", timeout=10)
```

3. Create a .sql file and write sql query in it & create a .batch file and write the following script in it.

```
SQLCMD -S SERVERNAME -d DATABASENAME -E -i "FOLDER PATH OF SQLQU
```

Findings of the Car Rental Data Analysis

Combined_Queries_24-10-26 200334.xlsx

1. Date Filtering

- Data was initially filtered for the period from **October 1, 2024, to October 31, 2024**.

2. Timestamp Conversion

- The UNIX timestamps in the **Created** column were converted to a human-readable date format.

3. Recent Data Extraction

- Filtered for the most recent data covering the **last month up to October 23, 2024**.

4. Week-on-Week Transaction Analysis

- Analysis focused on **average transaction amount** and **transaction count** on a weekly basis:
  - The **highest transaction volume** occurred in **Week 2**, followed by Weeks 4 and 1. An equal number of transactions were recorded in Weeks 3 and 5.
  - For average transaction amounts, **Week 3** had the highest, followed by Week 4 > Week 5 > Week 2 > Week 1.
- Additional columns, including **week start and end dates**, were added for clarity.

5. Financial Data Segmentation

- Data was filtered to include specific financial metrics: **Amount, Customer Earning, Merchant Amount, Stripe Fee**, and **Net Amount**. This allows for focused financial analysis.

6. Weekly Financial Summary

- Weekly summaries were generated with **total amount, customer earning, Stripe fee**, and **profit** metrics, providing insights into financial trends.

7. Total Amount vs. Transaction Count

- Weekly analysis comparing **total transaction amount** and **transaction count**:
    - **Highest transaction amount and count** were in **Week 2**, followed by Week 4. Week 3 had the **third-highest transaction amount** but only 7 transactions. **Week 1** recorded the lowest transaction amount.

8. Customer Earning vs. Transaction Count

- Weekly comparison of **total customer earnings** and **transaction count**:
    - Customer earnings were highest in **Week 2** and lowest in **Week 1**.

9. Stripe Fee vs. Transaction Count

- Weekly analysis of **total Stripe fees** compared to **transaction count**:
    - Stripe fees were highest in **Week 2**, followed by Week 4, Week 3, Week 5, and Week 1.

10. Profit Analysis by Week

- Weekly review of **total profit** and **transaction count**:
    - Profit peaked in **Week 2**, followed by Week 4, with the lowest profit in **Week 1**.

11. Refund Analysis

- Data was filtered to analyze cases involving **refunds**.

12. Country-Based Financial Analysis

- A breakdown of **total profit, amount, customer transactions,** and **Stripe fees** by country:

- **Singapore** generated the highest profit, with **Australia** in second. Singapore also had the highest total transaction amount, customer transactions, and Stripe fees.

13. City-Country Concatenation

- A new column combining **City and Country** was created for clarity. A table was also created with columns **ID, Amount, City_Country, Customer Transaction, Fee, Payout,** and **Date**.

14. City-Country Data Analysis

- **Sydney, Australia**, showed the highest transaction volume, total customer transaction, Stripe fee, and payout.

15. Weekly Financial Analysis for Brisbane, Australia

- Business activity in **Brisbane** occurred in **Weeks 1, 2,** and **5**. There was no business recorded in Weeks 3 and 4, requiring further investigation.

16. Weekly Financial Analysis for Jurong, Singapore

- Transactions occurred in **Weeks 1, 2, 4,** and **5**, with no business in **Week 3**.

17. Weekly Financial Analysis for Melbourne, Australia

- Transactions were recorded in **Weeks 2** and **4** only, with no business in **Weeks 1, 3,** and **5**.

18. Weekly Financial Analysis for Perth, Australia

- Business activity was observed in **Weeks 2** and **4**, with no transactions in **Weeks 1, 3,** and **5**.

Innovation

Company could add features like localized pricing and multilingual support to make it easier for customers in new regions, adding electric and hybrid vehicles would appeal to eco-friendly customers. Partnering with local rental agencies would also help increase vehicle options and local expertise. Finally, adding better

customer support and global insurance options would build trust, making the platform reliable and accessible for users worldwide.

Summary

This analysis of car rental data aims to increase profits for both the company and the customers who rent out their vehicles. The report shows key weekly patterns, including transaction volumes, customer earnings, Stripe fees, and profit margins. Singapore stands out as the top area for both transaction activity and profit. Weekly and regional insights provide a clear view of how revenue is spread and point out areas for improvement. Overall, the goal is to keep renters supplying vehicles, helping the company grow and build a profitable, sustainable business.