**SQL Querying Concepts**

Assume a table named “WebsiteAnalytics” which includes the following columns: **date**, **page\_views**, **conversions**, and **revenue**.

| **date** | **page\_views** | **conversions** | **revenue** |
| --- | --- | --- | --- |
| 1/3/2025 | 1200 | 30 | 450.5 |
| 2/3/2025 | 980 | 22 | 375.75 |
| 3/3/2025 | 1450 | 40 | 610.3 |
| 4/3/2025 | 1100 | 28 | 500.2 |
| 5/3/2025 | 890 | 18 | 320.9 |
| 6/3/2025 | 1300 | 35 | 540.75 |
| 7/3/2025 | 1500 | 42 | 620.8 |
| 8/3/2025 | 1250 | 33 | 505.6 |
| 9/3/2025 | 1100 | 27 | 470.25 |
| 10/3/2025 | 1600 | 50 | 700 |

Write out and explain the SQL queries you would use to calculate:   
1) the conversion rate

**Query**

SELECT

date,

(conversions \* 1.0 / page\_views) \* 100 AS conversion\_rate

FROM WebsiteAnalytics;

**Explanation**

* date: Selects the date column.
* (conversions \* 1.0 / page\_views): Divides the number of conversions by total page views to get the conversion rate as a decimal.
  + Multiplying by 1.0 ensures decimal division in SQL instead of integer division.
* \* 100: Converts the decimal to a percentage.
* AS conversion\_rate: Names the resulting column.

2) total revenue per day

**Query**

SELECT

date,

SUM(revenue) AS total\_revenue

FROM WebsiteAnalytics

GROUP BY date;

**Explanation**

* date: Groups the data by date.
* SUM(revenue): Adds up all revenue for the day.
* GROUP BY date: Required when using aggregation functions like SUM().
* AS total\_revenue: Names the resulting column.

3) average revenue per conversion

**Query**  
 SELECT

date,

(revenue \* 1.0 / conversions) AS avg\_revenue\_per\_conversion

FROM WebsiteAnalytics;

**Explanation**

* date: Selects the date column.
* (revenue \* 1.0 / conversions): Divides the total revenue by the number of conversions to calculate average revenue per conversion.
  + \* 1.0: Ensures decimal division.
* AS avg\_revenue\_per\_conversion: Names the resulting column.

**Python Scripting Concepts**

Your client has requested that you implement an email cleaning task, as some emails bounce when sent for various reasons. You are instructed to only clean emails that do not exist. Therefore, please write a Python script (including comments about your goals for each line of code) using a Jupyter notebook to produce a .csv file containing the non-existent emails (using a comma as the delimiter). Please refer to the attached **deliveries.csv** file. We recommend using the **Pandas** python library. You can deliver a .rar file with the scripts and the .csv or upload it to a GitHub repository (extra points for this last delivery method).

**Hint**: Utilize the **failure\_message** column to classify the non-existent bouncing emails.

**GitHub Repository**