Question 1 : Create a ranking of students based on score (highest first).

Answer:
```
SELECT
  student_id,
  name,
  course,
  score,
  RANK() OVER (ORDER BY score DESC) AS score_rank
FROM Student_Performance;
```

Explanation

Here, `RANK()` is used to give ranking based on the score in descending order.
The student with the highest score gets rank 1. If two students have the same score, they get the same rank.

Question 2 : Show each student's score and the previous student’s score (based on score order).

Answer:

```
SELECT
  student_id,
  name,
  score,
  LAG(score) OVER (ORDER BY score DESC) AS previous_score
FROM Student_Performance;
```

Explanation

The `LAG()` function is used to get the score of the previous student when ordered by score in descending order.
For the top scorer, the previous score will be `NULL` because there is no student above them.

Question 3 : Convert all student names to uppercase and extract the month name from join_date.

Answer:

```
SELECT
   student_id,
   UPPER(name) AS student_name,
   MONTHNAME(join_date) AS join_month
FROM Student_Performance;
```

Explanation

`UPPER()` converts all student names into uppercase letters.
`MONTHNAME()` extracts the month name from the `join_date`, making the date easier to understand.

Question 4 : Show each student's name and the next student’s attendance (ordered by attendance).

Answer:

```
SELECT
  student_id,
  name,
  attendance,
  LEAD(attendance) OVER (ORDER BY attendance) AS next_attendance
FROM Student_Performance;
```

Explanation

The `LEAD()` function is used to get the attendance of the next student when records are ordered by attendance.
For the last student in the order, the next attendance value will be `NULL`.

Question 5 : Assign students into 4 performance groups using NTILE().

Answer:

```
SELECT
   student_id,
   name,
   score,
   NTILE(4) OVER (ORDER BY score DESC) AS performance_group
FROM Student_Performance;
```

Explanation

`NTILE(4)` divides all students into 4 groups based on their scores.
Students are ordered by score from highest to lowest, and then evenly placed into 4 performance groups.

Question 6 : For each course, assign a row number based on attendance (highest first).

Answer:

```
SELECT
  student_id,
  name,
  course,
  attendance,
  ROW_NUMBER() OVER (
     PARTITION BY course
     ORDER BY attendance DESC
  ) AS row_num
FROM Student_Performance;
```

Explanation

Here, `ROW_NUMBER()` is used to give a unique number to each student within the same course. The numbering restarts for every course and is based on attendance from highest to lowest.

Question 7 : Calculate the number of days each student has been enrolled (from join_date to today). (Assume current date = '2025-01-01')

Answer:

```
SELECT
  student_id,
  name,
  join_date,
  DATEDIFF('2025-01-01', join_date) AS days_enrolled
FROM Student_Performance;
```

Explanation

`DATEDIFF()` calculates the number of days between the given current date and the student's join date.
This shows how long each student has been enrolled up to **1 January 2025**.

Question 8 : Format join_date as “Month Year” (e.g., “June 2023”).

Answer:

```
SELECT
  student_id,
  name,
  DATE_FORMAT(join_date, '%M %Y') AS formatted_join_date
FROM Student_Performance;
```

Explanation

`DATE_FORMAT()` is used to change the date format.
`%M` gives the full month name and `%Y` gives the year, so the output looks like “June 2023”.

Question 9 : Replace the city ‘Mumbai’ with ‘MUM’ for display purposes.

Answer:

```
SELECT
   student_id,
   name,
   REPLACE(city, 'Mumbai', 'MUM') AS city
FROM Student_Performance;
```

Explanation

`REPLACE()` is used to change the display value of the city.
Only the word **Mumbai** is replaced with **MUM**, and the original data in the table is not changed.

Question 10 : For each course, find the highest score using FIRST_VALUE().

Answer:

```
SELECT
  student_id,
  name,
  course,
  score,
  FIRST_VALUE(score) OVER (
     PARTITION BY course
     ORDER BY score DESC
  ) AS highest_score_in_course
FROM Student_Performance;
```

Explanation

Data is divided course-wise using `PARTITION BY`.
`FIRST_VALUE()` returns the highest score in each course because the data is ordered by score in descending order.