### 7. User-Defined Functions (UDFs):

#### a. Creating and Using UDFs:

User-Defined Functions (UDFs) allow you to define your own custom functions in programming languages like Python. In the context of SQL or data analysis tools, UDFs can be applied to manipulate and transform data.

Example 1: Creating a Simple UDF in Python

```python
# Python function to square a number
def square(x):
    return x ** 2

# Using the UDF on a list of numbers
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(square, numbers))

print(squared_numbers)
```

Example 2: Applying UDF in Pandas DataFrame

```python
import pandas as pd

# Sample DataFrame
data = {'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8]}
df = pd.DataFrame(data)

# Define a UDF to multiply each element by 2
def multiply_by_2(x):
    return x * 2

# Apply the UDF to a DataFrame column
df['A_doubled'] = df['A'].apply(multiply_by_2)

print(df)
```

#### b. Pandas UDFs:

Pandas UDFs (User-Defined Functions) allow you to apply custom functions to DataFrame columns efficiently using the `apply` method.

Example 1: Using Pandas UDF for Element-wise Operation

```python
import pandas as pd

# Sample DataFrame
data = {'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8]}
df = pd.DataFrame(data)

# Define a Pandas UDF to calculate the sum of two columns
@pd.api.extensions.register_series_accessor("custom")
class CustomAccessor:
    def __init__(self, series):
        self._validate(series)
        self.series = series

    def _validate(self, series):
        # Ensure that the series has the required data type
        if not pd.api.types.is_numeric_dtype(series.dtype):
            raise ValueError("Series must have numeric data type.")

    def sum_two_columns(self, other_column):
        return self.series + other_column

# Apply the Pandas UDF to create a new column
df['A_plus_B'] = df['A'].custom.sum_two_columns(df['B'])

print(df)
```

Example 2: Using Pandas UDF with Window Functions

```python
import pandas as pd

# Sample DataFrame
data = {'Category': ['A', 'A', 'B', 'B', 'A', 'A'],
        'Value': [1, 2, 3, 4, 5, 6]}
df = pd.DataFrame(data)

# Define a Pandas UDF to calculate the cumulative sum within each category
```

```
@pd.api.extensions.register_series_accessor("custom")
class CustomAccessor:
    def __init__(self, series):
        self.series = series

    def cumulative_sum_in_category(self):
        return self.series.groupby(df['Category']).cumsum()

# Apply the Pandas UDF with Window Function
df['Cumulative_Sum'] = df['Value'].custom.cumulative_sum_in_category()

print(df)
```

### 8. Window Functions:

#### a. Window Specification:

Window functions in SQL allow you to perform calculations across a specified range of rows related to the current row.

Example 1: Simple Window Function in SQL

```sql
SELECT
    employee_id,
    salary,
    AVG(salary) OVER () AS avg_salary
FROM
    employees;
```

Example 2: Using Window Function with PARTITION BY in SQL

```sql
SELECT
    department,
    employee_id,
    salary,
    AVG(salary) OVER (PARTITION BY department) AS avg_salary_by_department
FROM
    employees;
```

#### b. ROW_NUMBER, RANK, DENSE_RANK:

These are window functions used to assign a unique ranking to each row within a partition of a result set.

Example 1: ROW_NUMBER in SQL

```sql
SELECT
    employee_id,
    salary,
    ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num
FROM
    employees;
```

Example 2: RANK and DENSE_RANK in SQL

```sql
SELECT
    department,
    employee_id,
    salary,
    RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS rank,
    DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS
dense_rank
FROM
    employees;
```

Certainly! Let's explore more examples for each of the topics:

### 7. User-Defined Functions (UDFs):

#### c. Handling Multiple Input Parameters in UDF:

Example 3: UDF with Multiple Parameters in Python

```python
# Python function to calculate the area of a rectangle
def calculate_rectangle_area(length, width):
    return length * width

# Using the UDF with multiple parameters
```

```
length = 5
width = 3
area = calculate_rectangle_area(length, width)

print(f"The area of the rectangle is: {area}")
```

Example 4: Applying UDF with Multiple Parameters in Pandas DataFrame

```python
import pandas as pd

# Sample DataFrame
data = {'Length': [2, 4, 6, 8], 'Width': [3, 5, 2, 7]}
df = pd.DataFrame(data)

# Define a UDF to calculate the area of a rectangle with two columns
def calculate_rectangle_area(row):
    return row['Length'] * row['Width']

# Apply the UDF to create a new column for area
df['Area'] = df.apply(calculate_rectangle_area, axis=1)

print(df)
```

### 8. Window Functions:

#### b. Additional Window Functions - LEAD and LAG:

In addition to ROW_NUMBER, RANK, and DENSE_RANK, LEAD and LAG are window functions that allow you to access subsequent and previous rows in a result set, respectively.

Example 3: Using LEAD in SQL

```sql
SELECT
    employee_id,
    salary,
    LEAD(salary) OVER (ORDER BY salary DESC) AS next_salary
FROM
    employees;
```

Example 4: Using LAG with PARTITION BY in SQL

```sql
SELECT
    department,
    employee_id,
    salary,
    LAG(salary) OVER (PARTITION BY department ORDER BY salary DESC) AS
previous_salary
FROM
    employees;
```

These additional examples provide a broader understanding of User-Defined Functions, handling multiple parameters, and exploring more window functions like LEAD and LAG in the context of SQL. Feel free to adapt these examples to suit your specific needs!